#77 March 1983



($3.00 in Canada)

# Dr. Dobb's Journal

## For Users of Small Computer Systems

## AUGUSTA Part II

## A Small-C Operating System

## 6809 Threaded-Code

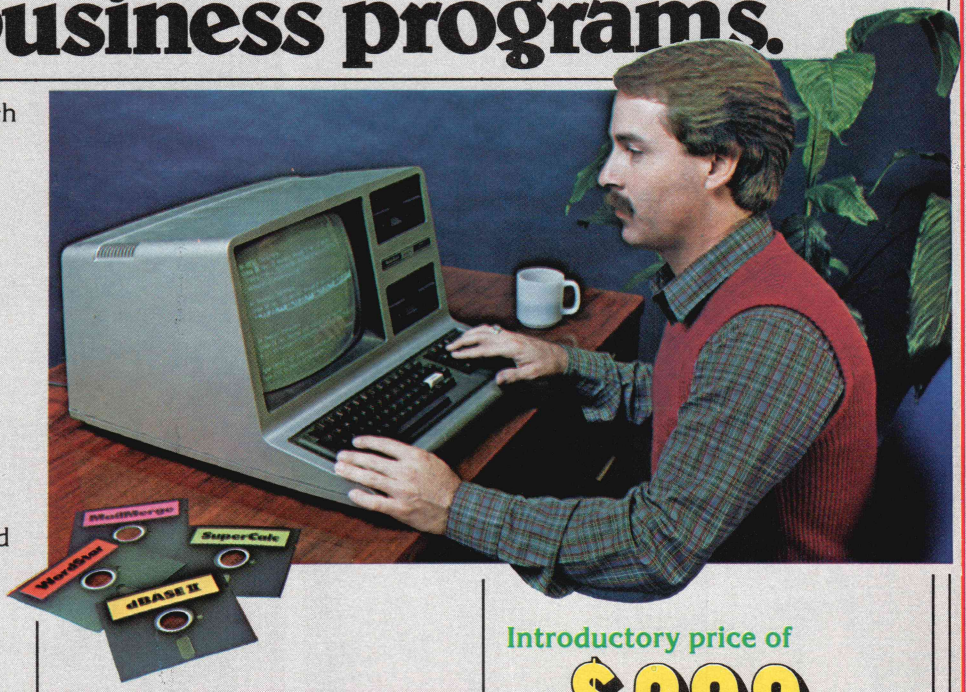
## Faster Smaller BASIC

# Put 64K CP/M® 2.2 in your TRS-80 Model III and tap into 2,000 business programs.

Now you can run programs such as WordStar, dBASE II, SuperCalc, MailMerge and virtually thousands of other CP/M-based programs on your TRS-80 Model III.

CP/M 2.2 is the industry standard operating system that gives you access right now to over 2,000 off-the-shelf business programs.

Our plug-in Shuffleboard III comes with 16K of RAM, giving your Model III the power of full 64K CP/M 2.2 without interference of the ROM or video memory. In fact, the Shuffleboard will appear transparent in the TRS-80 mode and will not interfere with any DOS operation.




## READ and WRITE Osborne, Xerox and IBM personal computer software plus many more popular formats.

Unfortunately, there is no standardized CP/M format for 5¼" diskettes. But we have developed a way to READ/WRITE and RUN standard programs under the following single-sided formats: Osborne 1 S/D, Xerox 820 S/D, IBM PC* D/D for CP/M 86 only, Superbrain D/D, Kapro II D/D, HP 125 D/D and TeleVideo D/D.

*Will Read and Write Only.

## Easy plug-in installation.

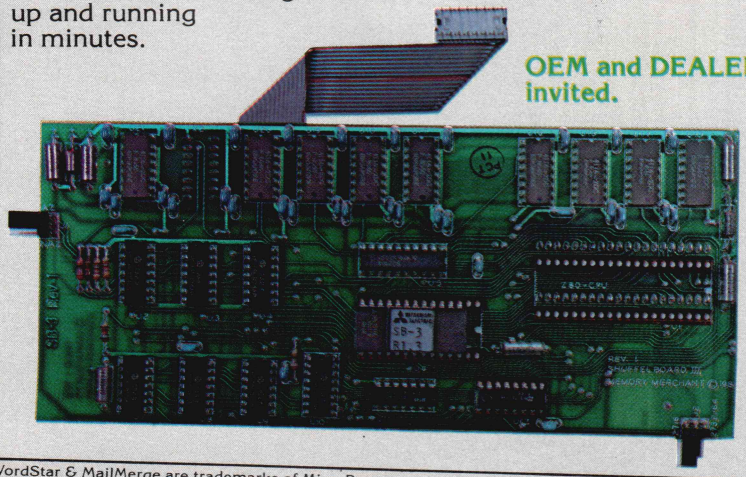
It's so simple. The Shuffleboard III plugs into two existing sockets inside your Model III. There are no permanent modifications, no cut traces and no soldering. You'll be up and running in minutes.

## New Products.

80 x 24 VIDEO BOARD: Features dual intensity screen, programmable cursor control for block, underline & blink rate, on-board bell with audible keyclick, battery-operated real time calendar/clock, full ASCII character set plus 256 special character graphics, dual RS-232 outputs and composite video output.

FLOPPY DISK CONTROLLER: Now you can access 5¼" and 8" floppy disk drives in any combination up to 4 drives of S/D density, S/D sided. Tap into a wealth of CP/M software which comes on 8" IBM 3740 format or Pickles & Trout CP/M for the Model II.

SOFTWARE: Additional CP/M software programs are available. Call or write for details.



## OEM and DEALER inquiries invited.

## Introductory price of $299.

The Shuffleboard III comes fully burned-in and tested complete with 64K CP/M 2.2 and MBASIC 80 interpreter, plus software manuals and a first class user's manual — with a 1-year limited warranty and 15-day no-risk free trial — for only $299.

## See the Shuffleboard III at your dealer's now.

Once you see what the Shuffleboard can do for your Model III you'll want one at once. If your dealer does not yet stock the Shuffleboard have him give us a call. Or send check, money order, VISA or MASTERCARD number (sorry, no COD's) plus $5 shipping per board ($17 outside the USA & Canada)* directly to the address below. Cal. residents please add sales tax. Credit card purchases can be phoned in directly and we'll ship from stock.

**(415) 483-1008**

*Air mail shipments to Canada & all other countries.

**Memory Merchant™**

**14666 Doolittle Drive San Leandro, CA 94577**
**(415) 483-1008**

WordStar & MailMerge are trademarks of MicroPro.
SuperCalc is a trademark of SORCIM.
dBASE II is a trademark of Ashton-Tate.
CP/M is a trademark of Digital Research.
TRS-80 is a trademark of Tandy Corporation.
IBM is a trademark of IBM Corporation.

Circle no. 2 on reader service card.

# Z-80® and 8086 FORTH

## PC/FORTH™ for IBM® Personal Computer available now!

**FORTH Application Development Systems** include interpreter/compiler with virtual memory management, assembler, full screen editor, decompiler, demonstration programs, utilities, and 130 page manual. Standard random access disk files used for screen storage. Extensions provided for access to all operating system functions.

| | |
|---|---|
| **Z-80 FORTH** for CP/M® 2.2 or MP/M | $ 50.00 |
| **8086 FORTH** for CP/M-86 | $100.00 |
| **PC/FORTH** for IBM Personal Computer | $100.00 |

**Extension Packages** for FORTH systems

| | |
|---|---|
| Software floating point | $100.00 |
| Intel 8087 support (PC/FORTH, 8086 FORTH only) | $100.00 |
| AMD 9511 support (Z-80, 8086 FORTH only) | $100.00 |
| Color graphics (PC/FORTH only) | $100.00 |
| Data base management | $200.00 |
| Symbolic Interactive Debugger (PC/FORTH only) | $100.00 |
| Cross Reference Utility | $ 25.00 |
| Curry FORTH Programming Aids | $150.00 |
| PC/GEN™ (custom character sets, IBM PC only) | $ 50.00 |

**Nautilus Cross-Compiler** allows you to expand or modify the FORTH nucleus, recompile on a host computer for a different target computer, generate headerless code, and generate ROMable code with initialized variables. Supports forward referencing to any word or label. Produces load map, list of unresolved symbols, and executable image in RAM or disk file. No license fee for applications created with the Cross-Compiler! Prerequisite: one of the application development systems above for your host computer.

Hosts: Z-80 (CP/M 2.2 or MP/M), 8086/88 (CP/M-86), IBM PC (PC/DOS or CP/M-86)
Targets: Z-80, 8080, 8086/88, IBM PC, 6502, LSI-11, 68000, 1802, Z-8

| | |
|---|---|
| Cross-Compiler for one host and one target | $300.00 |
| Each additional target | $100.00 |

| | |
|---|---|
| **AUGUSTA™** from Computer Linguistics, for CP/M 2.2 | $ 90.00 |
| **LEARNING FORTH**, by Laxen & Harris, for CP/M | $ 95.00 |
| **Z-80 Machine Tests** Memory, disk, console, and printer tests with all source code in standard Zilog mnemonics | $ 50.00 |

All software distributed on eight inch single density soft sectored diskettes, except PC/FORTH on 5¼ inch soft sectored single sided double density diskettes. Micropolis and North Star disk formats available at $10.00 additional charge.

Prices include shipping by UPS or first class mail within USA and Canada. Overseas orders add US$10.00 per package for air mail. California residents add appropriate sales tax. Purchase orders accepted at our discretion. No credit card orders.


## Laboratory Microsystems, Inc.

4147 Beethoven Street
Los Angeles, CA 90066
(213) 306-7412





Circle **no. 3** on reader service card.

*"THE RESULTS ARE IMPRESSIVE..."*
—Dennis Kitsz, 80 Microcomputing: 12/82

## Langley-St. Clair's* *Soft·View*™

# Replacement CRT's

## eliminates the strobe, flicker and fatigue from TRS-80's.™

Now you can upgrade your monitor with the new medium persistence green or amber phosphor tube.

State-of-the-art systems such as IBM™ and Apple III™ do not use the less costly "P4" B&W display tube because it is actually intended for TV viewing and its rapid strobes (60 times per second) cause irritating eye fatigue.

No amount of "green plastic" will solve this problem. But the new ***Soft·View*** CRT display tube from Langley-St. Clair will.

- Available in slow decay Green or medium decay "European Amber" (the standard in Europe)
- Made with Lead/Strontium impregnated glass that stops X-ray emission.
- Of high-contrast face glass that also stops most U.V. radiation.
- Available in frosted glass with extra Anti-Glare benefits.
- Easily installed...comes with pre-mounted hardware.
- Warranted for one full year against manufacturing defects or tube failure.
- The finest quality double-dark glass phosphor fields to produce dramatic contrast.
- Ideal for Word Processing and Programming, yet fast enough for Games and Graphics.

### LSIS *Soft·View*™ CRT'S

| | |
|---|---|
| ☐ #GN42 Green Phosphor | $79.95 |
| ☐ #GN42G Green Phosphor w/Anti-Glare | $89.95 |
| ☐ #OR34 Amber Phosphor | $89.95 |
| ☐ #OR34G Amber Phosphor w/Anti-Glare | $99.95 |
| also available: | |
| ☐ #R22G Red Phosphor w/Anti-Glare | $139.95 |
| ☐ #B22G Blue Phosphor w/Anti-Glare | $139.95 |

Plus: $7.00 for packing and UPS Shipping
$17.00 for Overseas, Parcel Post or UPS Blue Label
Add Sales Tax where applicable.
(Inquire about the CRT's we have available for many other computer models)

**For MasterCard and Visa Orders only, call**

**800/221-7070** (in N.Y. call 212/989-6876)

**Langley-St. Clair Instrumentation Systems, Inc.**
132 West 24th St., New York, N.Y. 10011


*World's largest supplier of upgraded replacement CRT's.



Circle **no. 4** on reader service card.

**Dr. Dobb's Journal**
For Users of Small Computer Systems
March 1983 Volume 8, Issue 3



**Subscription Rates:** $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

**Writer's Guidelines:** All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

**Donating Subscribers** **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

**Contributing Subscribers:** Thomas H. Grohne, G. V. Elkins, Robert M. Connors, William J. McEown, Robert C. Luckey, Robert N. Stabler, DeWitt S. Brown, John B. Palmer, Burks A. Smith, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Howard E. Decker, Jim Wilson, Frank Lawyer, Chris Pettus, Paul Lutus, Ben Goldfarb, Rodney Black, Steven Weiss, Pat Phelan, John Brodie, James Fleming, Unison World, Thomas Davis, Jan Steinman, Ronald E. Johnson, G. Hunter, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr., Ted A. Reuss III, Infoworld, Stan Veit, Western Material Control, S.P. Kennedy. **Sustaining Subscribers:** Pete Roberts, Steven Fisher Enterprises. **Lifetime Subscriber:** Michael S. Zick.

**Foreign Distributors** **UK & Europe:** Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014 Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian distributors)*

# CONTENTS

## ARTICLES

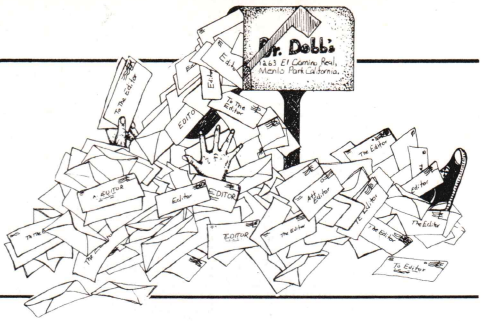


## DEPARTMENTS



*Dr. Dobb's Journal* (USPS 307690) is published twelve times per year by People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, CA 94025. Second class postage paid at Menlo Park, California 94025 and additional entry points. Address correction requested. Postmaster: send form 3579 to Box E, Menlo Park, California 94025 · 415/323-3111

# LETTERS




## Off to a Good Start

Dear Editor,

Your January issue was stimulating as usual. Of the dozen or so computer journals that arrive every month, *Dr. Dobb's* is one of the few that I always read. The good Dr. Dobb deserves particular thanks for introducing me to the Forth language several years ago.

On the whole I disagree strongly with Tom Pittman's January letter. But he does have a valid point with regard to those interminable listings. Everybody has a modem these days. Perhaps you should consider putting the program files on Compuserve or The Source or an in-house dial-up line. Even at 300 baud, it sure beats typing.

I enjoyed Ray Duncan's discussion of the numeric evaluation of trigonometric functions. As usual, he is brief and cogent, but on one point his trigonometry is a little bit rusty. We do not have to extract a square root to obtain the cosine. The cosine is simply the sine shifted by 90 degrees. Thus cos(x)=sin(x+90) or for radian phreaks, cos(x)=sin(x+PI/2).

To the entire staff at *Dr. Dobb's*, I wish a successful and prosperous new year. Keep up the good work.

Sincerely,
Joseph McDermott
265 Bogert Road
River Edge, NJ 07661

## A Slightly Slicker Fix for Small-C

Dear Editor,

Mr. Macpherson was kind enough to send me a copy of his letter to you (*DDJ* #76, pp. 10-11) describing a Small-C bug involving *continue* statements within *switch* statements. He was quick to fathom the problem and find a solution. After studying his patch, I settled on a slightly more efficient and straightforward solution. I hope he won't mind if I recommend this patch over his own.

First, in the function *doswitch* below the call to *addwhile* (line 184, page 39, *DDJ* #74), insert the line

```
*wqptr + WQLOOP - WQSIZ) = 0;
```

Then change the following functions to read as shown. Modified lines begin with a # character.

(page 42, *DDJ* #74)

```
    dobreak() {
      int *ptr;
#   if ((ptr=readwhile(wqptr))==0)
            return;
    modstk ((ptr[WQSP]), NO);
    jump(ptr[WQEXIT]);
    }

    docont() {
      int *ptr;
#   ptr = wqptr;
#   while (1) {
#     if ((ptr=readwhile(ptr))==0) return;
#     if (ptr[WQLOOP]) break;
#     }
    modstk((ptr[WQSP]), NO);
    jump(ptr[WQLOOP]);
    }
```

(page 45, *DDJ* #74)

```
    delwhile() {
#     if (wqptr > wq)
            wqptr=wqptr-WQSIZ;
      }

#   readwhile(ptr) int *ptr;{
#     if (ptr <= wq) {
#       error("out of context");
        return 0;
        }
#     else return (ptr-WQSIZ);
    }
```

This will compile with the original compiler, so those bringing it up for the first time with this patch should have no problem.

The response to my article has been most encouraging. I was surprised to find that a number of well known microcomputer companies are using the little compiler. I am looking forward to seeing more software contributions written for Small-C.

Sincerely,
J. E. Hendrix
Rt. 1, Box 74-B-1
Oxford, MS 38655

## Augusta Short Circuits

Dear Dr. Dobb,

I was very pleased to see the article, "Augusta," by Edward Mitchell. Ada™ is unfortunately "doomed to succeed," so it is good to see that those of us who live on small machines will be able to see all the reasons why it is being panned by such notables as Hoare ("The Emperor's Old Clothes," *Communications of the ACM*, Vol. 24, No. 2, February 1981). I will also be interested in seeing a compiler written in BASIC, of all things. Unfortunately, Mr. Mitchell left out one of the most useful features of Ada, the package.

However, Mr. Mitchell states that the "short circuit" conditionals of Ada are unusual. I beg to differ. Many modern languages, such as C and LISP, support short circuit evaluation of conditionals. Most languages (including, notably, Pascal) leave the question of whether conditionals short circuit up to the implementor. In fact, one of the popular Fortran compilers for the IBM 370 would test Mr. Mitchell's example [if (n<>0) and (j/n=1) then] by testing whether or not j/n=1, and if that were false, "short circuit" the test n<>0. This state of affairs means that the defensive (i.e., smart) programmer will act as if things will *not* be short circuited, and also not put statements with side effects in conditionals, just in case things *are* short circuited. After working with such things, it is truly enjoyable to use something which lets you know what's going on – whether it be C, LISP, or Ada.

Mike Meyer
Box 1749
Norman, OK 73070

## Compliments for Kossow

Dear Dr. Dobb,

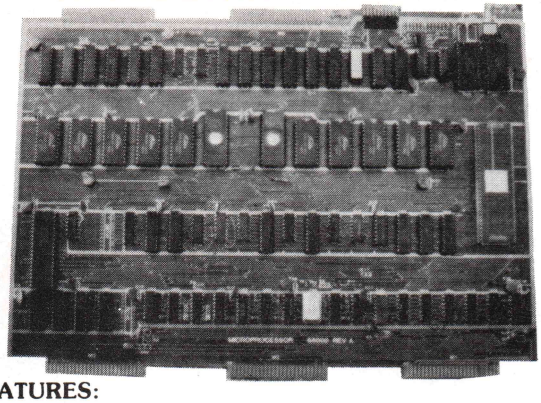
Thanks for presenting Allen Kossow's 68000 cross-assembler. Mr. Kossow has done an excellent job of managing the large instruction set and numerous addressing modes of this new processor.

In the past I have made use of system software listings provided by the doctor. This time, however, because of the imminent arrival of a 68000-based development system at my workplace, I took advantage of Mr. Kossow's offer. Twenty-five dollars is a reasonable copying fee in my opinion.

I was pleased to find that the single-density RT-11 diskette contained an executable file in addition to the Fortran and Macro source files. Because of this, I immediately set to testing. Five problems were discovered using examples from *68000 Assembly Language Programming* by Kane, Hawkins and Leventhal (Osborne/McGraw Hill, 1981).

The first hitch was an illegal instruction trap encountered when run on a PDP-11/04. It was traced to the Macro routine file, specifically, the ASH instruction in routine GETBIT. Replacing this one instruction with four (4) ASR R0 instructions cleared things up nicely. If you are using a DEC processor with EIS

# *NEW!* M-68000
## SINGLE BOARD COMPUTER



**FEATURES:**

**16 bit Motorola 68000 CPU operating at 5 MHz or 10 MHz, 20K of on board fast static RAM, 16K bytes of on board EPROM space, 7 autovectored interrupts, 3 memory/device expansion buses, 2 serial communication ports (RS-232 C), 16 bit bidirectional parallel port, 5-16 bit counter/timers with vectored interrupt and time of the day clock. On board monitor allows to download and debug programs generated on APPLE II, TRS-80 and CP/M using our M68000 Cross Assembler.**

**PRICE:**

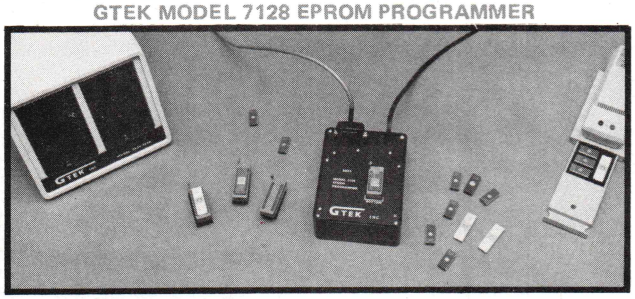
| | |
|---|---|
| **M68K Bare board with documentation** | **$ 99.95** |
| **M68MON monitor & mapping PROM's** | **$135.00** |
| **M68000-6 CPU** | **$ 95.00** |
| **M68K Parts Kit** | **$249.00** |
| **M68000 Cross Assembler** | **$125.00** |
| **M68K Documentation only** | **$ 15.00** |
| **Shipping & handling (Domestic)** | **$ 3.50** |
| **(foreign)** | **$ 15.00** |

**CALIFORNIA RESIDENTS ADD 6% TAX**

**EMS** Educational Microcomputer Systems
P.O. BOX 16115, IRVINE, CA 92713-6115
(714) 553-0133

Circle **no. 5** on reader service card.

## DEVELOPMENT HARDWARE/SOFTWARE
### GTEK MODEL 7128 EPROM PROGRAMMER



- Microprocessor based intelligence for ease of use and interface. You send the data, the 7128 takes care of the rest.
- RS-232 interface and ASCII data formats make the 7128 compatible with virtually any computer with an RS-232 serial interface port.
- Auto-select baud rate.
- Use with or without handshaking. Bidirectional Xon/Xoff supported. CTS/DTR supported.
- Devices supported as of DEC 82.

| NMOS | NMOS | CMOS | EEPROM | MPU'S |
|---|---|---|---|---|
| 2758 | 2508 | 27C16 | 5213 | 8748 |
| 2716 | 2516 | 27C32 | X2816 | 8749 |
| 2732 | 2532 | C6716 | 48016 | 8741 |
| 2732A | 2564 | 27C64 | | 8742 |
| 2764 | 68766 | | | 8751 |
| 27128 | 8755 | | | 8755 |

- Read pin compatible ROMS also.
- Automatic use of proper program voltage based on type selected.
- Menu driven eprom type selection, no personality modules required. (40 pin devices require adapter)
- INTEL, Motorola and MCS-86, Hex formats. Split facility for 16 bit data-paths. Read, program, and formatted list commands also.
- Interupt driven type ahead, program and verify real time while sending data.
- Program single byte, block, or whole eprom.
- Intelligent diagnostics discern between eprom which is bad and one which merely needs erasing.
- Verify erasure and compare commands.
- Busy light indicates when power is being applies to program socket.
- Complete with TEXTOOL zero insertion force socket and integral 120 VAC power supply. (240 VAC/50HZ available also)
- High Performance/Cost ratio.

●●● **Model 7128 PRICE $389.00** ●●●

**MODEL 7128 SOCKET ADAPTERS**
MODEL 481 allows programming of 8748, 8749, 8741, 8742 single chip processors.
**Price $98.00**

MODEL 511 allows programming the 8751, Intel's high powered single chip processor.
**Price $174.00**

MODEL 755 allows programming the 8755 EPROM/IO chip
**Price $135.00**

**MODEL 7128/24** - budget version of the 7128. Supports 24 pin parts thru 32K only. Upgradable to full 7128 capacity.
**Price $289.00**

Non-expandable, very low cost models available for specific devices.
MODEL 7128-L1 for 2716 only **$149.00**
MODEL 7128-L2 for 2732 only **$179.00**

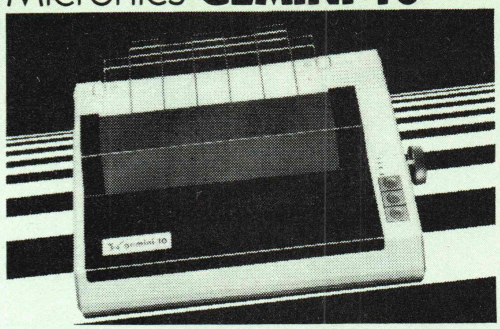
Also available from stock:

| | |
|---|---|
| Eprom Erasers UVP model DE-4 | $78.00 |
| Avocet Systems Cross Assemblers | $200.00 |
| RS-232 Cable Assemblies | $25.00 |
| Programmable Devices | call |
| Complete development systems | $3240.00 |

GTEK INC.
Post Office Box 289
Waveland, Mississippi 39576
(601) 467-8048

Circle **no. 6** on reader service card.

## Star Micronics **GEMINI-10**



**$419.88 UPS DELIVERED**

- 100 characters per second, bi-directional, logic-seeking printhead action (48 lines/min.) with 2K print buffer, expanable to 4K on-board
- 9 x 9 matrix produces proportional, 10, 12, 17 cpi with true descenders, double width, double strike, italics, & special graphics characters
- 120 x 144 hi-resolution dot-addressable graphics matrix
- Subscripts, superscripts, underlining, backspace, plus 2K user-programmable character ROM, perf skip, vert/horz tabs
- Friction/tractor standard; handles 3-part forms (8.5")

### PRINTERS

| | |
|---|---|
| Anadex DP-9500A | $1459.88 |
| Anadex DP-9501A | $1459.88 |
| Anadex DP-9629A | $1549.88 |
| Centronics 122-1 | $829.88 |
| Centronics 122-3 | $949.88 |
| Centronics 352 | $1649.88 |
| Centronics 353 | $2324.88 |
| C. Itoh Prowriter | $499.88 |
| w/RS-232C | $609.88 |
| C. Itoh Prowriter 2 | $734.88 |
| w/RS-232C | $789.88 |
| C.Itoh F 10 Starwriter, 40 cps Parallel or RS-232C | $1499.88 |
| C.Itoh F-10 Printmaster, 55 cps Parallel or RS-232C | $1799.88 |
| F-10 Tractor | $289.88 |
| Daisywriter 2000 | $1089.88 |
| Daisywriter Tractor | $149.88 |
| Daisywriter Cable | $49.88 |
| Diablo 620 | $1269.88 |
| Diablo 630 | $1969.88 |
| Diablo 630 KSR | $2694.88 |
| 630 Tractor | $314.88 |
| DMP-85 Printer | $469.88 |
| IDS Microprism | $679.88 |
| IDS Prism 80 | $1104.88 |
| Prism 80 w/graphics | $1339.88 |
| Prism 80 w/sheetfeed | $1459.88 |
| Prism 80 w/4-color | $1539.88 |
| IDS Prism 132 | $1269.88 |
| Prism 132 w/graphics | $1339.88 |
| Prism 132 w/sheetfeed | $1459.88 |
| Prism 132 w/4-color | $1699.88 |

### PRINTERS

| | |
|---|---|
| Microline 80 | $349.88 |
| Microline 82A | $439.88 |
| 80/82A Tractor | $59.88 |
| 82A Roll Paper Holder | $49.88 |
| Microline 83A | $694.88 |
| 82A/83A Okigraph ROM | $44.88 |
| Microline 84 w/graphics & tractor Parallel, 200 cps | $1044.88 |
| RS-232C, 200 cps | $1164.88 |
| NEC PC-8023A | $509.88 |
| NEC 3510 | $1929.88 |
| NEC 3530 | $1809.88 |
| NEC 3550 | $2199.88 |
| 3500 Tractor | $239.88 |
| Qume Sprint 9/45 | $2109.88 |
| Smith Corona TP-1 10 or 12 cpi, parallel or RS-232C | $599.88 |

CALL FOR PRICES on Epson, DIP, MPI, Datasouth, & other printers.

### MODEMS

| | |
|---|---|
| Hayes 300 Baud | $239.88 |
| Hayes 1200 Baud | $569.88 |
| Hayes Micromodem-II w/Software | $299.88 |
| | $329.88 |
| Novation Apple Cat 1200 | $579.88 |
| Novation Apple Cat 300 | $334.88 |
| Novation 1200 Upgrade | $324.88 |
| Novation Auto Cat 300 | $224.88 |
| Novation Auto-Cat 1200 | $569.88 |
| Novation Cat | $159.88 |
| Novation D Cat | $159.88 |
| Signalman Mark 1 | $89.88 |

**Monitors, cables, IBM PC & Apple/Franklin hardware & software also available. Call for more information.**

**Information & Orders**
**(603)-673-8857**

**Orders Only: (800)-343-0726**

**NO HIDDEN CHARGES**

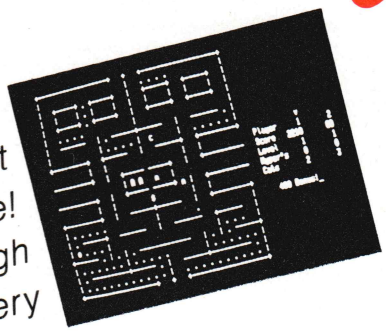
FREE UPS shipping on all orders—No extra charge to use credit cards—All equipment shipped factory fresh with manufacturer's warranty—COD orders accepted ($10 fee) No purchase orders accepted—No foreign or APO orders accepted—Minimum $50 per order—This ad prepared in November: prices subject to change.

HIGH TECHNOLOGY AT AFFORDABLE PRICES
**THE BOTTOM LINE**
MILFORD NH 03055-0423

Circle **no. 7** on reader service card.

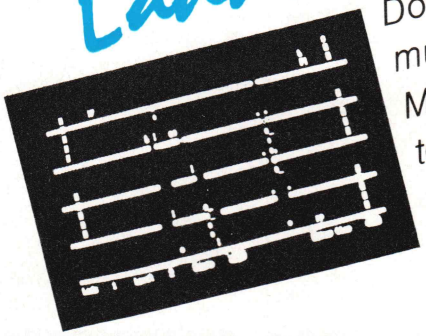
# Arcade Style Games for CP/M

## Catchum™



Hungry little Cs chased through a maze by Ms and As, eating energy dots as they go, just like the little yellow guys you know and love! Multiple levels and screens! Keeps track of high scores! Easily personalized for virtually every terminal! And no quarters needed!

## Ladder™



Dodge falling barrels as you climb to the top of this multi-level course! Just like the game with the gorilla! Multiple levels and screens! Personalized for your terminal in less than a minute! Enjoy the excitement of the popular arcade games with your CP/M computer!

Each **$24.95**

Both **$39.95**

VISA®

Please include $2 postage and handling with each order. California residents include 6% sales tax (6½% in L.A. County). Indicate 5¼" or 8" disk, single or double density, single or double sided, soft, 10 or 16 sectors. Call or write for complete catalogue with challenging word games like ADVENTURE and STAR TREK ADVENTURE and other entertaining and useful programs. Our offices are at 2463 McCready Avenue, Los Angeles, CA 90039. Our telephone number is (213) 661-2031.

MasterCard

**CDE CALIFORNIA DIGITAL ENGINEERING**

**P.O. BOX 526 ★ HOLLYWOOD, CA 90028**

Circle **no. 8** on reader service card.

(Extended Instruction Set) like an LSI-11/23 or LIS-11/34, then this change is unnecessary.

The second problem showed up as a reversal of the source and destination registers in all ADDX, SUBX, ABCD, and SBCD instructions. A fix is as follows:

Replace these three lines in subroutine PRCESS

```
1910 IF(OP1EA.EQ.5)
       OPSKEL=OPSKEL+8
     OPSKEL=OPSKEL+OP2DA
     OBJBUF(1)=
      OPSKEL+(OP1DA*"1000)
```

by inserting these three lines

```
     OPSKEL=OPSKEL+8
1910 OPSKEL=OPSKEL+OP1DA
     OPJBUF(1)=OPSKEL+
      (OP2DA*"1000)
```

The third bug prevented any MOVE instruction from being assembled as MOVEQ(quick move). A fix follows:

Replace these four lines in subroutine PRCESS

```
     IF(IMODE.NE.3)GOTO 304
     IF(OPNWRD(3).EQ.0) GOTO
      301
     IF(OPNWRD(3).EQ.-1) GOTO
      301
     GOTO 304
301 . . .
```

by inserting the following three lines

```
     IF(IMODE.NE.0) GOTO 304
     IF(OPNWRD(3).EQ.0) GOTO
      301
     IF(OPNWRD(3).NE.-1) GOTO
      304
301 . . .
```

This causes the assembler to optimize a MOVE immediate instruction to the MOVEQ form only if the datum size is defaulted and the datum is in the legal range for MOVEQ(-129 < arg < 128). For a consistent treatment of quick mode instructions, the following change should be made to make ADDQ and SUBQ operate in the same manner:

insert this line into PRCESS

```
     IF(IMODE.NE.0) GOTO 536
```

in between these two existing lines

```
     IF(OPNFLG.EQ.1) GOTO 536
```

and

```
     IF(OPNWRD(2).GE.1.AND.
      OPNWRD(2).LE.8) GOTO 550
536 . . .
```

Problem number four prevented assembly of the CMPM form of the CMP instruction. A fix is below:

Replace this line in subroutine PRCESS

```
400 . . .
     IF((OP1EA.EQ.5) .AND.
      (OP2EA.EQ.5)) GOTO 480
```

by this new line

```
     IF((OP1EA.EQ.4) .AND.
      (OP2EA.EQ.4)) GOTO 480
```

A fifth problem was noted in assembly of the LEA instruction where the addressing mode is register indirect with index. An example is LEA -1(A0,D0.W), A1. I have not had an opportunity to analyze the problem any deeper.

My compliments to Allen Kossow. A piece of software of this size seldom appears with so few glaring errors. Keep up the good work.

Sincerely,
Steve Albrecht
Tracor Northern Instruments
Middleton, WI 53562

Dear Editor,

I have an ulterior motive for writing. I *must* have one of Al Kossow's (*DDJ* No. 68, "Multi-68000s in a Personal System") computers.

I've written to Mr. Kossow to express my enthusiasm. A project like his is an ambitious one so I didn't really expect an answer.

Yes, I know he said any future developments would be published in *DDJ*. The point of *this* letter is to urge *DDJ* to publish the info . . .

Anxiously awaiting each and every issue . . .

Judd Ellmers
1A Rolling Ridge Road
Montvale, NJ 07645

*Editor's note: We hope to do an update soon. In the meanwhile, can any* DDJ *readers help us out on multi-68000s?*

## More on Graphics Algorithms . . .

Gentlemen:

The vector generation algorithm published in the December issue of *DDJ* is not new. Several years ago, I purchased a Houston Instrument plotter and the manual included an implementation in BASIC of what was essentially that algorithm. I found the algorithm described in one of the BYTE Books called *Bits and Pieces*. There it was credited to somebody at IBM. Most implementations reduce the calculation per step even further than Mr. Michalski's version. Usually F is initialized to 2*DY-DX and in the loop, F is tested relative to zero as the first step of the loop. If F is negative, only X is incremented before the new point is plotted and F=F+2*DY is calculated. If F is zero or positive, both X and Y are incremented before the new point is plotted and F=F+2*DY-2*DX is calculated beforehand so only one addition per step is necessary to update F. Here I have used the same notation as Mr. Michalski except for using D instead of delta. Well, it is a good idea no matter how many times it is reinvented.

This is the simplest and probably the most useful of what I call the where-do-we-go-from-here algorithms. The idea is that the choice of the next point to be moved to or set is limited to one of the eight points next to the last point. This limitation may be inherent in the graphics device or imposed to insure the drawing of a continuous line. Knowledge of the general direction of the curve is used to limit the choice to two of these points, and some easily calculated error function is used to choose between the two. So far this straight line algorithm and the one for doing circles and arcs are the only ones for which I have worked out the details and successfully implemented. I did both of them in BASIC first and then finding this a bit slow, I resorted to a mixture of Fortran and assembly language. With that I was able to keep my plotter going at close to its maximum speed. Although I have not yet worked out the details, it appears that these techniques could be extended to any of the conic sections, but at a price of increasing complexity and decreasing speed.

Another graphics related matter that I have been planning to write you about is related to Mr. Taylor's ellipsoid problem which appeared in the Clinic. I was one of those that sent in a solution to his problem. Ever since his response to the solutions appeared in the August issue, I have been thinking about it and I have reached the conclusion that he posed the wrong problem. He talks of perspective projection and yet the problem that he posed was projection parallel to the Z axis which is orthographic projection. I have worked out the solution for perspective projection, but since it involves even more calculation than the solution for orthographic projection, which he found to involve too much calculation for his purposes, it probably will be of little use to him.

Sincerely yours,
David S. Tilton
27 Pennacook Street
Manchester, NH 03104

## . . . And Variations on Michalski's

People:

I was interested in your simple vector generation algorithm in the December *DDJ* – interested because I had devised the same underlying algorithm for use in a mailing list application.

A businessman has a mailing list with 90 names (for example). He wishes to send some promotional material, but to only 20 people (small budget, maybe). Therefore he wants a way of extracting 20 names from the list of 90 in an even distribution.

Hopefully, you can see that selecting 20 names from 90 is a similar problem to

drawing a vector which rises 20x points and runs 90y points.

The algorithm used in the mailing list application was: Let y run from 0 to 89, and set x=y*20div90. X will increase slowly from 0 to 19. For each input name (y), print it out only if the value of x has increased from the previous y (integer arithmetic). In Pascal:

```
var
  x, y, xprev : integer;
begin
  xprev := -1;
  for y := 0 to 90-1 do begin
      getname;
      x := y * 20 div 90;
      if x <> xprev then
        putname;
      xprev := x
  end
end
```

I hope this shows the idea. Obviously a useful program would use variables in place of 90 and 20, and probably would have to make an initial pass through the list to count how many names are there. The flowchart published in the article uses more code but is more efficient because it eliminates the multiply and divide operations.

An amusing variation: The above program tries to print 20 names, but if the original list only had 8 names, it would only print 8. By replacing the "if" statement with a "for" loop, you can guarantee 20 printed names no matter how small the mailing list is (provided it's not empty).

```
. . .
getname;
x := y * 20 div 8;
for xprev := xprev to x-1 do
      putname;
xprev := x
. . .
```

With this program, each of the 8 names is printed two or three times to give an output list of 20 entries.

Yours truly,
Peter Raynham
10 Camrose Crescent
Scarborough, Ontario
Canada M1L 2B6

## Forth Cosine with 8087

Dear Editor:

The purpose of this letter is twofold. (1) I wish to share with your readers my enthusiasm toward Forth Inc.'s method of structuring 8087 floating point in their Forth for the IBM PC. (2) Since trigonometric functions are not included in Polyforth (at least not in the version I own), I would like to put into public domain a program (see Listing 1, this page) for calculating cosines. Besides being fast (470 microseconds) and accurate (16 digits), the algorithm I developed is unusual because it uses an unspecified 8087 feature and involves no conditional branching. Thus, the algorithm should be of interest to non-Forth programmers.

Forth is not a popular floating-point language and has no floating-point standards. Traditionally, floating-point numbers have been stored with integers on the parameter stack which resides in memory. This is somewhat inefficient since there are no operations between the two numerical types and one tends to obstruct the other, thereby requiring extensive stack manipulation in a floating-point program.

Consider the 8087, a chip with a stack of its own comprised of eight 80-bit registers that can perform most internal operations faster than a 64-bit number can be transferred to memory. Indeed, the 8087 can swap two 80-bit numbers on its stack faster than the 8088 can swap two 16-bit numbers in memory. Clearly, with the 8087, it would be a serious waste of time to automatically transfer 8-byte floating-point numbers to a memory stack and then back again between operations.

Forth Inc. has made a rational deviation from tradition by simply using the 8087 stack as an extra Forth stack. It wouldn't surprise me if this chip-language combination is unbeatable among microcomputers for speed. To test my hypothesis, I invite benchmark challenges, particularly from owners of 68000-based computers. I only require that the program be a practical 64-bit number cruncher such as matrix multiplication and not an abstraction.

Regarding the cosine algorithm, from the angle, V, the 8087 instruction FPTAN returns the sides of a right triangle, X(V) and Y(V). Intel literature states the restriction, "0<V<PI/4"; however, I discovered that my 8087 was completely accurate in the range -PI/2<V<PI/2. Using this feature, I came up with the following algorithm that is best described in equation form:

```
Let U = Angle
S1*[FABS( FABS[ FPREM
( U,2*P1 ) ] - PI ) - PI/2]
          ( U,2*PI ) ] - PI ) - PI/2 ]
COS(U) = Y(V) / FSQRT( X(V)*X(V)
          + Y(V)*Y(V) )
```

Notice that multiplying by S1 keeps V from exceeding its limits. I have included an implementation of this algorithm using Polyforth assembler mnemonics. The sine function can be obtained by simply subtracting PI/2 from U.

Sincerely yours,
Steven A. Ruzinsky
2110 S. Austin Blvd.
Cicero, IL 60650

**(See Listing 1 below)**

## Subscription Glitch

Dear Doctor Dobb,

I'm sure you have heard stories like mine many times before, but I would be grateful if you eyeballed my output.

I used to be a small microprocessor with very underdeveloped software. When I went to the beach, big strong mainframes would kick sand in my face and attractive young minis would laugh at me.

How delighted was I when I discovered what a dose of *DDJ* did for my development! So wonderful were the results, that it is little surprise that I soon became addicted to monthly shots of *DDJ*. Naturally when my subscription renewal became due, I raced down to the bank at my fastest baud rate, obtained a bank cheque in U.S. dollars for two years (yes, I wasn't going to risk cold turkey), and sent it off airmail immediately.

Can you imagine my alarm when some months later another renewal notice

**Listing 1.**
**8087 Cosine Routine in Polyforth.**

```
254
 0 ( 8087 Cosine in Polyforth, 470 microsec. )
 1
 2         LVARIABLE S1
 3         .9999999999999999 S1 L!
 4
 5 CODE
 6   COS   FLD1 FLDPI FSCALE 1 N) FSTP 1 FXCH
 7         FPREM 1 N) FSTP FABS FLD1 FCHS FLDPI
 8         2 N) FSUB -POP FSCALE 1 N) FSTP
 9         1 FXCH FABS 1 N) FSUB REV R64 S1 FMUL
10         FPTAN 1 N) FLD 0 N) FMUL NO 1 FXCH
11         0 N) FMUL NO 1 N) FADD FSQRT 1 N) FDIV
12         NEXT
13
14
```

arrived! Of course I straightaway photocopied the bank cheque duplicate and sent it off with an impassioned plea.

Now (fear and trembling) I have been told my subscription has expired because the cheque was not received. Bless my chips, what will I do? I know it cannot be a computer error because we never err. To make matters worse, my bank tells me it will be several weeks before they can find out whether the bank cheque was banked at the other end.

Please, Doctor Dobbs, help a poor deprived little processor whose last shot of *DDJ* was November 1982! Please scan your files to make sure whether my cheque has arrived. If my bank tells me the cheque is lost, I will send another as fast as I can.

Yours faithfully,

(signed for Neil's computer by)

Dr. Neil Trezise
Wild Dog Creek Road
St. Andrews
Victoria 9761
Australia

*Ed. Note: Those of you who have experienced similar problems, please bear with us. During our recent change of data services, a number of our sheep became separated from the fold. We are working hard to get this corrected, but unfortunately these things take a bit of time. Sorry for the inconvenience. Just let us know (most of you already have) and we will make the correction.*

## Standard Deviation

Hi guys,

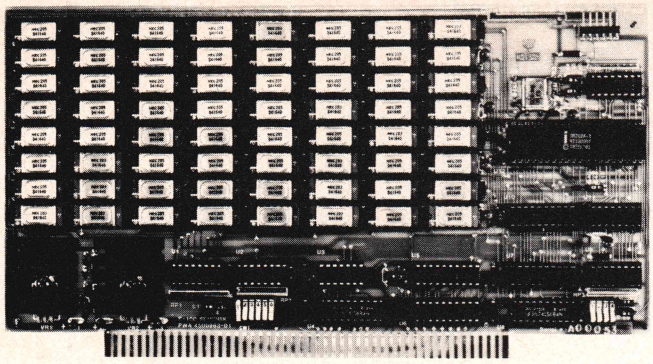
Good to see that C compiler is progressing; nothing better than a good free software. I'd like to say a few things about C pertaining to portability (a very important issue).

I've been seeing more and more C code that does not follow the guidelines published by Kernighan & Ritchie (*The C Programming Language*). That's a real no-no. If everyone adheres to those guidelines with care, C will remain a very portable language. J. E. Hendrix is guilty of the crime himself in his December article. Since publication of the compiler implementation and its libraries is a restatement of the standard, it should be done with careful attention to detail.

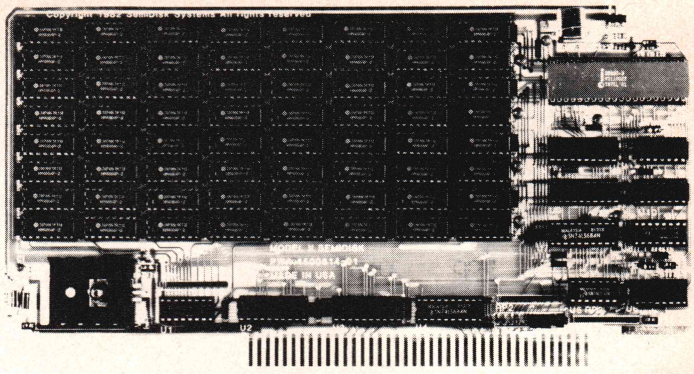
Hendrix describes several functions in his auxiliary library which do not maintain the spirit of the K & R teachings (the C bible and de facto definition). His dtoi, decimal to integer, conversion function sounds almost the same as K & R's atoi (ASCII to integer). Doesn't matter which name sounds better – atoi is the standard. Also atoi only requires one parameter;

*(Continued on page 90)*

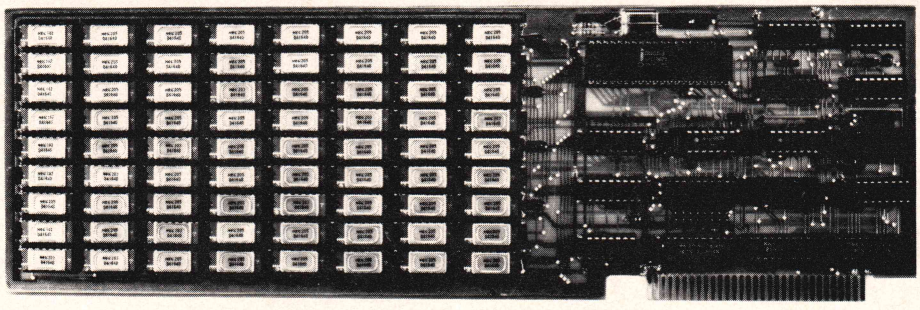
# A FULL LINE OF SEMIDISKS



**S-100**



**TRS 80 Model 2**



**IBM Personal Computer**

Do you use your computer? Or does your computer "use" you? Face it, if you're using floppies, your time is being wasted. Because a floppy is an inefficient random access storage device. Each time the processor wants to transfer data, it has to wait an eternity for the disk to rotate and the head to move.

So what do you do? Get a SemiDisk, quick. It's a large capacity semiconductor memory board that is driven by software to operate like a disk drive. Without all the waiting. Do everything you'd do on a floppy or hard disk, with no modifications to your software or hardware. Two board sizes are available: 512K and 1 Megabyte. (the highest density microcomputer memory board in the world) And you can put up to 8 megabytes in a system by adding more storage boards.

What do you need to use it? Just an S-100 system with CP/M 2.2. Or a TRS-80 Model 2 system with CP/M 2.2. Or an IBM Personal Computer. That's it. No special processors, DMA, I/O, or disk controllers are required. Plug it in and run the installation program, and you're on your way. Fast! Even better, we supply full source code to the driver software, in case you'd like to do your own interfacing.

Best of all, the SemiDisk's price won't warp your wallet. Compare specs, cost/megabyte, storage capacity, and compatibility with the competition. You'll see that the SemiDisk is a disk emulator truly worthy of the name. SemiDisk has battery-backup capability, too.

Consider our limited warranty: A full year, covering all parts and labor. Consider our liberal 15 day return policy. Price? $1995 for 512K byte SemiDisk, $2995 for 1 Megabyte SemiDisk. Both from stock. $10.00 for manual. VISA, Mastercard, COD orders accepted. Dealer and OEM inquiries welcomed. (Specify system type and disk format when ordering.)

**Someday, you'll get a SemiDisk.**
**Until then, you'll just have to . . . . . wait.**


T.M.

**SemiDisk SYSTEMS**

P.O. Box GG
Beaverton, OR 97075

**(503)-642-3100**

Call (503)-646-5510 for CBBS®/NW, a Semi-Disk-equipped computer bulletin board.
SemiDisk trademark of SemiDisk Systems; TRS-80 trademark of Radio Shack

Circle **no. 11** on reader service card.

# You are invited to become a member of the ASSOCIATION FOR COMPUTING MACHINERY...

As one who is involved with computers, you owe it to yourself to join the major technical organization in the field — the Association for Computing Machinery (ACM). Whether your involvement is professional or recreational, you have a vital "need to know." ACM, the quintessential computer information source, can fill this need through its:

• *Ten major journals — you receive* ***Communications of the ACM,*** *one of the most respected publications in the field, free with your member dues.* • *32 Special Interest Groups (SIGs) — from personal computing (SIGPC) to small systems (SIGSMALL), each publishing a Newsletter on a particular computing discipline listed below.* • *40+ conferences sponsored annually by ACM and its SIGs.* • *100+ Chapters sponsoring Professional Development Seminars and hosting the ACM Lectureship Program.*

PLUS, the opportunity to make numerous professional contacts that can help you in your work.

There's an old saying that goes, "you're known by the company you keep." If you join ACM, you join ranks with the best!

**Join now and we'll send you, *free-of-charge,* a copy of the Proceedings of the 5th Symposium on Small Systems.**

## ACM Membership Application

### MEMBER CLASSES

**Voting Member:** You must a) subscribe to the purposes of ACM; b) have attained professional stature as demonstrated by intellectual competence and ethical conduct in the arts and sciences of information processing; and c) have earned a Bachelor's Degree or academic equivalent, or have 4 years full time experience in information processing. A Voting Member may vote and hold office in ACM.

**Associate Member:** You must subscribe to the purposes of ACM. Associate Members have the same privileges and benefits as Voting Members except the right to vote and hold office.

**Student Member:** You must be registered in an accredited educational institution on a full-time basis.

Payment must accompany application. SEND TO: H

**Association for Computing Machinery**
P.O. Box 12114, Church Street Station
New York, NY 10249

Name _____

Address _____

_____

City/State/ZIP _____

### DUES: Circle appropriate dues.

| | |
|---|---|
| Voting/Associate Members | $40.00 |
| Members of IEEE-CS (receive a $5 discount) | 35.00 |
| Members of the following overseas computing societies, ACS, AICA, BCS, CIPS, HKCS, IPA, NGI (receive an $8 discount). Overseas residents: See "Notes" section | 32.00 |
| Student Members | 13.00 |
| Student Member with $5 dues credit. Students who subscribe to *Journal of the ACM, Computing Surveys,* or *Computing Reviews* are entitled to a $5 dues credit. If you wish to subscribe to any one of the above, circle the $8 dues and the appropriate subscription rate for the journal selected in the "Publications" section | 8.00 |

### PUBLICATIONS: Circle appropriate rate(s)

| | |
|---|---|
| **Computing Surveys** (quarterly) 103 | $10.00 |
| **Journal of the ACM** (quarterly) 102 | 10.00 |
| **Computing Reviews** (monthly) 104 | 16.00 |
| **Collected Algorithms,** Initial Vols. I, II, III & 1 yr's quarterly updating supplements 105 | 75.00 |
| **Transactions on** (all quarterlies) | |
| **Mathematical Software/TOMS** 108 | 18.00 |
| **Database Systems/TODS** 109 | 18.00 |
| **Programming Languages and Systems/TOPLAS** 110 | 18.00 |
| **Graphics/TOG** 112 | 24.00 |
| **Office Information Systems/TOOIS** 113 scheduled for 1/83 | 20.00 |
| **Computer Systems/TOCS** 114 scheduled for 2/83 | 20.00 |

### SPECIAL INTEREST GROUPS (SIGs): Circle appropriate rate(s)

| | Voting/ Associate | Student |
|---|---|---|
| **SIGACT** (Automata & Computability Theory) 001 | $ 2.50 | $ 2.50 |
| **SIGAPL** (APL) 032 | 5.00 | 4.00 |
| **SIGARCH** (Computer Architecture) 002 | 20.00 | 10.00 |
| **SIGART** (Artificial Intelligence) 003 | 10.00 | 6.00 |
| **SIGBDP** (Bus. Data Process. & Mgmt.) 004 | 7.50 | 5.00 |
| **SIGBIO** (Biomedical Computing) 005 | 14.00 | 5.00 |
| **SIGCAPH** (Computers & Physically Handicapped, Print) 006 | 10.00 | 5.00 |
| **SIGCAPH** (Cassette Edition) 029 | 10.00 | 5.00 |
| **SIGCAPH** (Both Print & Cassette) 030 | 14.00 | 9.00 |
| **SIGCAS** (Computers & Society) 007 | 8.00 | 4.00 |
| **SIGCHI** (Computer & Human Interaction) 026 | 10.00 | 6.00 |
| **SIGCOMM** (Data Communication) 008 | 12.00 | 9.00 |
| **SIGCPR** (Computer Personnel Research) 010 | 8.00 | 4.00 |
| **SIGCSE** (Computer Science Education) 011 | 11.00 | 5.00 |
| **SIGCUE** (Computer Uses in Education) 012 | 10.00 | 7.00 |
| **SIGDA** (Design Automation) 013 | 3.00 | 3.00 |
| **SIGDOC** (Documentation) 033 | 12.00 | 2.00 |
| **SIGGRAPH** (Computer Graphics) 015 | 10.00 | 5.00 |
| **SIGIR** (Information Retrieval) 016 | 6.00 | 3.00 |
| **SIGMAP** (Mathematical Programming) 018 | 10.00 | 7.50 |
| **SIGMETRICS** (Measurement & Evaluation) 019 | 9.00 | 3.00 |
| **SIGMICRO** (Microprogramming) 020 | 10.00 | 6.00 |
| **SIGMOD** (Management of Data) 014 | 3.00 | 3.00 |
| **SIGNUM** (Numerical Mathematics) 021 | 11.00 | 5.50 |
| **SIGOA** (Office Automation) 027 | 7.50 | 3.00 |
| **SIGOPS** (Operating Systems) 022 | 8.00 | 4.00 |
| **SIGPC** (Personal Computing) 035 | 7.00 | 5.00 |
| **SIGPLAN** (Programming Languages) 023 | 22.00 | 11.00 |
| **SIGPLAN-AdaTEC** (Tech. Comm. on Ada) 037 | 15.00 | 10.00 |
| **SIGPLAN-FORTEC** (Tech. Comm. on Fortran) 038 | 6.00 | 3.00 |
| **SIGSAC** (Security, Audit & Control) 036 | 12.00 | 4.00 |
| **SIGSAM** (Symbolic & Algebraic Manipulation) 024 | 7.50 | 3.00 |
| **SIGSIM** (Simulation) 025 | 5.00 | 2.00 |
| **SIGSMALL** (Small Computing Systems and Applications) 031 | 9.00 | 4.00 |
| **SIGSOFT** (Software Engineering) 034 | 6.00 | 4.00 |
| **SIGUCCS** (University and College Computing Services) 028 | 10.00 | 5.00 |

### CERTIFICATION: Fill in if applicable.

**Voting Member Applicants**—You must satisfy at least one of the following requirements and sign below.

1. Bachelor's Degree. Institution: _____

2. Equivalent level of education. Institution: _____

_____

_____

3. Four full time years of experience (attach statement)

I attest the above is correct. _____
SIGNATURE

**Student Member Applicants**—Your Faculty Advisor must certify your full-time status.

_____
EDUCATIONAL INSTITUTION

_____
FACULTY ADVISOR'S SIGNATURE DATE

**Joint Membership Applicants**—You must indicate your affiliation, member # and sign below. Only one discount is permitted

_____
SIGNATURE AFFILIATION MEMBER #

### PURPOSES OF ACM & SIGNATURE

1. To advance the sciences and arts of information processing including, but not restricted to, the study, design, development, construction, and application of modern technology, computing techniques and appropriate languages for general information processing, storage, retrieval, transmission/communication, and processing of data of all kinds, and for the automatic control and simulation of processes.

2. To promote the free interchange of information about the sciences and arts of information processing both among specialists and the public in the best scientific and professional tradition.

3. To develop and maintain the integrity and competence of individuals engaged in the practice of information processing.

I hereby affirm that I subscribe to the purposes of ACM (as indicated above) and understand that my membership is not transferable. I enclose a check, bank draft or money order in the full amount.

_____
SIGNATURE DATE

**Total Amount: $_____**

**Notes For Overseas Members:** If you'd like to join ACM please write ACM Headquarters for an Overseas Membership Application. Mention this issue of *DDJ* and you'll still receive a free book.

| FOR OFFICE USE ONLY | Member No. | St/Cntry | Zip Code/City | T/C | R/C | M/C |
|---|---|---|---|---|---|---|
| | | | | | | |

**Association for Computing Machinery**
11 West 42nd Street, New York, NY 10036 • (212) 869-7440 • Telex: 421686

CUT APPLICATION ALONG DOTTED LINE

Circle **no. 12** on reader service card.

# 16-BIT SOFTWARE TOOLBOX

**by Ray Duncan**

## Cromemco's Dual CPU

One of the most pleasant fringe benefits of starting up a software house is that it gives one the opportunity to buy all sorts of beautiful new hardware for "Research and Development." When Cromemco announced their combination Z-80 and 68000 "DPU" board early this year, I was fascinated but wary. The board seemed as though it potentially could give the same impetus to the 68000 among personal computer users as the Godbout Dual CPU board gave the 8088: providing a safe, known development environment to fall back on (Z-80 and CP/M-80), while giving ready access to the power of the 68000 microprocessor. However, I was a bit skeptical because most of Cromemco's boards in the past have not completely followed the S-100 bus standard, and were difficult to mix and match with components produced by other vendors. In addition, Cromemco's software is expensive and incompatible with standard CP/M application packages. Of course, being a computer junkie, I ordered the board anyway.

About three months after I put down a deposit, the board finally arrived and immediately got stuck on the shelf for another two months due to the pressure of other work. Finally, during the December lull, I found time to drag the DPU out and look it over. The component is built on a typical S-100, silk-screened, multilayer printed circuit board, and displays the usual solid Cromemco design and construction. It contains about sixty integrated circuits including the familiar looking Z-80 and the monstrous looking 64-pin 68000. In a departure from Cromemco's previous practice, all components except the two microprocessors and three other support ICs are wave soldered directly to the board instead of being mounted on sockets. The board is designated as "Revision E" in the silk-screened legend, but it still has no less than eight little green wires snaking their way around – this leads one to believe that the design of this board is not too stable yet.

The DPU board automatically starts up in the Z-80 mode of operation. When a 1 is output to port 0FFH, the DPU switches to the 68000 mode. The first time the 68000 is used, it obtains its stack pointer from location 0 and its program counter from location 4. To give control back to the Z-80, the 68000 must write a zero to location 0FFFFFFH (its I/O is memory mapped). Subsequently, when either processor resumes execution, the contents of its registers and instruction pointer are unchanged from the previous invocation. Since there is no direct communciation between the registers of the two microprocessors, information must be passed back and forth by temporarily saving it in memory. Listing 1 shows how to switch between the two microprocessors under software control.

I can guess what you're thinking if

you've read this far. In these times of economic unease, few of us can afford to junk our 8-bit systems and shell out several thousand dollars for a complete complement of Cromemco boards including memory and disk controller in order to get started with a workable 68000 system. But I have good news for you. I was tipped off by a friend that the Dual CPU board was "more S-100 compatible" than previous Cromemco products. In a moment of rashness, I set the jump-on-restart address for the Z-80 to 0000 by cutting two jumpers, dropped the DPU board into my Compu-Pro Z-80 system, and turned on the power. The system booted up and ran perfectly!

This opens up enormous possibilities for experimentation with the 68000. You can do all of your software development with a familiar CP/M-80 support structure for disk and terminal I/O. When CP/M-68K becomes available, you can write your initial BIOS and cold boot loader under CP/M-80, just as many of you did when the Godbout 8085/8088 CPU board was first released.

By the way, you will find *68000 Assembly Language Programming* by Gerry Kane *et al.* to be an extremely useful reference work as you are getting started. This book is devoted entirely to describing the instruction set from a programmer's point of view and is filled with helpful examples; it does not waste any space describing the hardware – for this you can refer to the manufacturer's literature.

I should note in passing that Compu-Pro has also started delivering their 68000 CPU. It costs only $625 in contrast to the $1000 for the Cromemco board; however, it doesn't have the advantage of the on-board Z-80. Compu-Pro is also selling a stand-alone 68000 Forth system for $200, but is not yet delivering the Digital Research operating system.

## More on Memory Tests

Some readers will recall that when I reviewed the Microsoft RAMCARD for the IBM Personal Computer, I had some critical things to say about the memory test program that came with it. At the time, I was just a bit disgusted with the program's style of interaction with the operator but had no reason to find fault with the actual usefulness of the program.

I must report that the program has a much more serious defect than I imagined: if the board it is testing has read errors causing parity errors, the test program crashes completely! Instead of getting the dandy display of the error type and location as the manual specifies, the parity error causes a hardware interrupt forcing transfer to the ROM BIOS, the message "PARITY ERROR2" is displayed on the screen, and the system goes dead. Apparently the test program does not reset the

### Listing 1.

**Switching between the Z-80 and the 68000 on Cromemco's DPU board.**

```
; switching on the 68000
;
;
            ld      hl,init     ;move initial values
            ld      de,0        ;for stack pointer and
            ld      bc,8        ;program counter into
            ldir                ;locations 0 and 4
                                ;
            ld      a,1         ;now turn on the 68000
            out     0ffh,a      ;

; switching on the Z-80
; (i/o ports are memory mapped in the top)
; (64 k of the address space on the 68000)
;
            move.b  #0,0ffffffh
;
```

FREE with software purchase — One CPM Handbook

# DISCOUNT SOFTWARE

✓ = New items

**ASHTON-TATE**
dBASE II . . . . call for price ($4??)

## CP/M®

**ARTIFICIAL INTELLIGENCE®**
Medical(PAS-3) . . . . $849
Dental (PAS-3) . . . . $849

**ASYST DESIGN®/FRONTIER**
Prof Time Accounting . . . . $549
General Subroutine . . . . $269
Application Utilities . . . . $439

**DIGITAL RESEARCH®**
CP/M 2.2®
NorthStar . . . . $149
TRS-80 Model II (P+T) . . . . $159
Micropolis . . . . $175
CP/M-Intel MDS . . . . $135
✓ PL/1-80 . . . . $449
BT-80 . . . . $179
MAC . . . . $ 85
RMAC . . . . $179
Sid . . . . $ 65
Z-Sid . . . . $ 90
Tex . . . . $ 90
DeSpool . . . . $ 49
CB-80 . . . . $459
CBasic-2 . . . . $ 98
Link-80 . . . . $ 90

**FOX & GELLER**
Quickscreen . . . . $135
Quickcode . . . . $265
dutil . . . . $ 65

**MICRO-AP®**
S-Basic . . . . $269
Selector IV . . . . $295
Selector V . . . . $495

**MICRO DATA BASE SYSTEMS®**
HDBS . . . . $269
MDBS . . . . $795
DRS or QRS or RTL . . . . $269
MDBS PKG . . . . $1999

**MICROPRO®**
✓ WordStar . . . . $279
Customization Notes . . . . $449
Mail-Merge . . . . $ 99
WordStar/Mail-Merge . . . . $369
DataStar . . . . $249
WordMaster . . . . $119
SuperSort I . . . . $199
✓ Spell Star . . . . $139
CalcStar . . . . $259

**MICROSOFT®**
✓ Basic-80 . . . . $199
Basic Compiler . . . . $329
Fortran-80 . . . . $349
Cobol-80 . . . . $589
M-Sort . . . . $175
Macro-80 . . . . $144
Edit-80 . . . . $ 84
MuSimp/MuMath . . . . $224
MuLisp-80 . . . . $174
FPL: Bus. Planner . . . . $595

**ORGANIC SOFTWARE®**
TextWriter III . . . . $111
DateBook II . . . . $269
Milestone . . . . $269

**OSBORNE® (McGraw/Hill)**
General Ledger . . . . $ 59

**SAVE $255 ON PRODUCTIVITY PAC #3!**

Everything you need: a wordprocessor, spreadsheet and database. And a phenomenally low, low price!

| | Retail | Regular Discount |
|---|---|---|
| Final Word | $300 | $270 |
| Plannercalc | $ 99 | $ 50 |
| Condor I | $295 | $275 |
| | $694 | $595 |

**SPECIAL COMBINATION PRICE: $439**
Offer good to the end of the month of publication of this magazine. Call for our other PAC prices.

Acct Rec/Acct Pay . . . . $ 59
Payroll w/Cost . . . . $ 59
All 3 . . . . $129
All 3 + CBASIC-2 . . . . $199
Enhanced Osborne (vandatta) (Includes CBasic) . . . . $269

**PEACHTREE®**
General Ledger . . . . $399
Acct Receivable . . . . $399
Acct Payable . . . . $399
Payroll . . . . $399
Inventory . . . . $399
Surveyor . . . . $399
Property Mgt . . . . $799
CPA Client Write-up . . . . $799
P8 Version . . . . Add $234
MagiCalc . . . . $269
Other . . . . less 10%

**STAR COMPUTER SYSTEMS**
✓ G/L, A/R, A/P, Pay . . . . $ 349
All 4 . . . . $1129
Legal Time Billing . . . . $ 849
Property Mngmt . . . . $ 849

**STRUCTURED SYSTEMS®**
Business Packages, Call for Price

**SORCIM®**
✓ SuperCalc . . . . $249
Trans 86 . . . . $115
Act . . . . $157

**SUPERSOFT®**
Ada . . . . $270
Diagnostic I . . . . $ 49
Diagnostic II . . . . $ 84
Disk Doctor . . . . $ 89
Forth (8080 or Z80) . . . . $149
Fortran . . . . $219
Fortran w/Ratfor . . . . $289
C Compiler . . . . $225
Star Edit . . . . $189
Scratch Pad . . . . $266
StatsGraph . . . . $174
Analiza II . . . . $ 45
Dataview . . . . $174
Disk Edit . . . . $ 89
Encode/Decode II . . . . $ 84
Optimizer . . . . $174
Super M List . . . . 68
Term II . . . . $179
Zap Z-8000 . . . . $450
Utilities I . . . . $ 54
Utilities II . . . . $ 54

**ACCOUNTING PLUS**
1 Module . . . . $385
4 Modules . . . . $1255
All 8 . . . . $4500

**UNICORN®**
Mince . . . . $149
Scribble . . . . $149
Both . . . . $249
The Final Word . . . . $270

**WHITESMITHS®**
"C" Compiler . . . . $600
Pascal (incl "C") . . . . $850

**"PASCAL"**
✓ Pascal/MT+ Pkg . . . . $429
Compiler . . . . $315
Sp Prog . . . . $175
Pascal/Z . . . . $349
Pascal/UCSD 4.0 . . . . $670
Pascal/M . . . . $355
Tiny Pascal . . . . $ 76

**"DATA BASE"**
FMS-80 . . . . $894
dBASE II . . . . $595
✓ Condor I . . . . $275
Condor II . . . . $535
FMS-81 . . . . $445

**"WORD PROCESSING"**
WordSearch . . . . $179
✓ SpellGuard . . . . $199
Peachtext . . . . $289
Magic Spell . . . . $269
Spell Binder . . . . $349
Select . . . . $495
The Word . . . . $ 65
The Word Plus . . . . $145
Palantier-I (WP) . . . . $385

**"COMMUNICATIONS"**
Ascom . . . . $149
BSTAM . . . . $149
BSTMS . . . . $149
Crosstalk . . . . $139
Move-it . . . . $ 89

**"OTHER GOODIES"**
Micro Plan . . . . $419
Plan 80 . . . . $269
Target (Interchange) . . . . $125
Target (Planner) . . . . $189
Target (Task) . . . . $299
Plannercalc . . . . $ 50
Tiny "C" . . . . $ 89
Tiny "C" Compiler . . . . $229
Nevada Cobol . . . . $179
MicroStat . . . . $224
Vedit . . . . $130
MiniModel . . . . $449
StatPak . . . . $449
Micro B+ . . . . $229
Raid . . . . $224
String/80 . . . . $ 84
String/80 (source) . . . . $279
ISIS CP/M Utility . . . . $199
Lynx . . . . $199
Supervyz . . . . $ 95
ATI Power . . . . $ 75
Mathe Magic . . . . $ 95
CIS COBOL . . . . $765
ZIP MBASIC, CBASIC . . . . $129
Real Estate Analysis . . . . $116

## APPLE II®

**BRODERBUND**
G/L (with A/P) . . . . $444
Payroll . . . . $355

**INFO UNLIMITED®**
EasyWriter (Prof) . . . . $155
Datadex . . . . $129
EasyMailer (Prof) . . . . $134
Other . . . . less 15%

**MICROSOFT®**
✓ Softcard (Z-80 CP/M) . . . . $239
Fortran . . . . $179
Cobol . . . . $499
Tasc . . . . $139
✓ Premium Package . . . . $549
✓ RAM Card . . . . $129

**MICROPRO®**
✓ Wordstar . . . . $199
MailMerge . . . . $ 99
Wordstar/MailMerge . . . . $349
SuperSort I . . . . $159
Spellstar . . . . $129
CalcStar . . . . $175
DataStar . . . . $265

**VISICORP®**
✓ Visicalc 3.3 . . . . $189
Desktop/Plan II . . . . $219
Visiterm . . . . $ 90
Visidex . . . . $219
Visiplot . . . . $180
Visitrend/Visiplot . . . . $259
Visifile . . . . $219
Visischedule . . . . $259

**PEACHTREE®**
G/L, A/R, A/P, Pay or Inventory (each) . . . . $224
Peach Pack P40 . . . . $795

**SOFTWARE DIMENSIONS, INC.**
Accounting Plus II, G/L, AR, AP, or Inventory (each) . . . . $385
(Needs G/L to run)

**"OTHER GOODIES"**
Super-Text II . . . . $127
Data Factory . . . . $134
DB Master . . . . $184
Versaform VS1 . . . . $350
VH1 . . . . $445

## 16-BIT SOFTWARE

**WORD PROCESSING**
IBM PC
✓ Wordstar . . . . $279
✓ Spellstar . . . . $175
Mailmerge . . . . $109
Easywriter . . . . $314
Easyspeller . . . . $159
Select/Superspell . . . . $535
Write On . . . . $116
Spellguard (also available for 8" 8086 systems) . . . . $229
SP Law (for Spellguard) . . . . $115
Textwriter III . . . . $189
Spellbinder . . . . $349
Final Word . . . . $270

**LANGUAGE UTILITIES**
**IBM PC**
Crosstalk . . . . $174
BSTAM . . . . $149
BSTMS . . . . $149

**8" 16-BIT SYSTEMS**
✓ Pascal MT+ /86, SSP . . . . $679
CBasic 86 . . . . $294
Pascal M/86 . . . . $445
Act 86 . . . . $157
Trans 86 . . . . $115
XLT 86 . . . . $135

**16-BIT 8" AND DISPLAYWRITER**
CP/M 86 . . . . $294
MP/M 86 . . . . $585

**OTHERS**
**IBM PC**
SuperCalc . . . . $269
VisiCalc . . . . $219
Easyfiler . . . . $359
Mathemagic . . . . $ 89
CP/M Power . . . . $ 65
Condor 21 . . . . $265
Condor 22 . . . . $535
Condor 23 . . . . $895
Condor 20Q . . . . $175
Condor 20R . . . . $265
Statpak . . . . $449
Optimizer . . . . $174
Desktop Plan II . . . . $219
Desktop Plan III . . . . $259
Visidex . . . . $219
Visitrend . . . . $259
Many others available for use with the "Baby Blue Board®"

**8" 16-BIT SOFTWARE**
SuperCalc . . . . $269
CP/M Power . . . . $ 65

## FORMATS AVAILABLE:

8" single density
8" OSI
Superbrain
Micropolis/Vector Graphic
NorthStar Horizon
NorthStar Advantage
Osborne
Heath/Zenith
Cromemco
Televideo
Xerox 820
Dynabyte
Hewlett-Packard 125
NEC
Eagle
Apple II/III
Otrona
TRS-80 Model I/II/III
DEC VT-180
Altos
CP/M-86
IBM PC

# LOWER PRICES, COME HELL OR HIGH WATER.

ORDERS ONLY • CALL TOLL FREE • VISA • MASTERCHARGE

**U.S. 1-800-421-4003 • CALIF. 1-800-252-4092**

Outside Continental U.S.—add $10 plus Air Parcel Post • Add $3.50 postage and handling per each item • California residents add 6½% sales tax • Allow 2 weeks on checks. C.O.D. $3.00 extra • Prices subject to change without notice. All items subject to availability • ®—Mfgs. Trademark. Blue Label $3.00 additional per item.

CP/M is a registered trademark of DIGITAL RESEARCH, INC.

**THE DISCOUNT SOFTWARE GROUP**

6520 Selma Ave. Suite 309 • Los Angeles, Ca. 90028 • (213) 837-5141

Int'l TELEX 499-0446 DISCSOFT LSA • USA TELEX 194-634 (Attn: 499-0446)
TWX 910-321-3597 (Attn: 499-0446)

Circle **no. 16** on reader service card.

interrupt transfer vector so that it can intercept these events and perform the appropriate error analysis.

## Regarding 8086/88 Segment Registers

The Intel 8086/88 documentation is rather vague about the use of default segment registers in the various indirect addressing modes. The *iAPX 86,88 User's Manual* states on page 2-12 that "most variables (memory operands) are assumed to reside in the current data segment, although a program can instruct the Bus Interface Unit to access a variable in one of the other currently addressable segments. . . ."

Leo Scanlon, the author of a book on 8088 assembly-language programming (which is in press at the Robert J. Brady Co.), has written to point out a surprising exception to this rule. The base indexed addressing modes [BP+SI] or [BP+DI], which I had always assumed were used to extract data from multi-dimensional data arrays, do *not* use DS (Data Segment register) as the default but rather form their address in combination with the SS (Stack Segment register). Teleologically, this is a bit unexpected, but Leo proved it experimentally with the program in Listing 2. Table 1 is derived from a similar table in Leo's new book and is reprinted with his permission as an aid to *DDJ* readers.

## IBM Low Resolution Graphics Revisited

In an earlier column I printed a BASIC listing which demonstrated how to initialize and use the low resolution color graphics mode on the IBM Personal Computer, allowing the display of sixteen color pixels with a resolution of 160 by 100. James Murphy, of Madison, Wisconsin, has kindly sent in a set of Forth routines to plot points in the lo-res mode (see Listing 3); these execute about twenty times faster than the original BASIC version even though they are written in high-level code.

In Screen #75 of the listing, the definitions "TR" and "CRT_LRC_INIT" take care of setting up the video controller chip's various registers. "CLEAR_BUFF" initializes the video memory map with the required values so that each pixel will be displayed as a solid block of color. "LRCG" (mnemonic for Low Resolution Color Graphics) uses the previously defined words to initialize the display and prepare for plotting. In Screen #76, "!DOT" (Store Dot) accepts an address and a color and sets the appropriate pixel in the memory map. X must be in the range 0-159 and Y in the range 0-99, with (X,Y)=(0,0) being the upper left corner of the screen. Finally, Screen #77 contains a nice little demonstration program which draws a test pattern of nested squares in different colors.

Readers who wish to extend the low resolution graphics capability even further can refer to the Forth line drawing routine which was printed in the July 1982 issue of *DDJ*.

(Listing 3 on page 18)

DDJ

| Addressing Mode | Operand Format | Segment Register |
|---|---|---|
| Register | reg | none |
| Immediate | data | none |
| Direct | disp | DS |
| | label | DS |
| Register indirect | [BX] | DS |
| | [BP] | SS |
| | [DI] | DS* |
| | [SI] | DS |
| Base relative | [BX+disp] | DS |
| | [BP+disp] | SS |
| Direct indexed | [DI+disp] | DS |
| | [SI+disp] | DS |
| Base indexed | [BX+SI+disp] | DS |
| | [BX+DI+disp] | DS |
| | [BP+SI+disp] | SS |
| | [BP+DI+disp] | SS |

*In string instructions, the destination address is formed by the combination of DI and ES, and the ES register cannot be overoverridden.

**Notes:**

1. "disp" is optional for base indexed addressing.
2. "reg" can be any 8- or 16-bit register, except IP.
3. "data" can be an 8- or 16-bit constant value.
4. "disp" can be an 8- or 16-bit signed displacement.

**Table 1.**
**8088 addressing modes and segment register defaults.**

**Listing 2.**
**Leo Scanlon's program to determine the default segment registers for base indexed addressing on the Intel 8086/88.**

```
STACK   SEGMENT  PARA STACK 'STACK'
        DB       100 DUP(0FFH)
STACK   ENDS
DATA    SEGMENT  PARA 'DATA'
        DB       0ABH

DATA    ENDS
CODE    SEGMENT  PARA 'CODE'
PROG    PROC     FAR

        ASSUME   CS:CODE,DS:DATA,SS:STACK
        MOV      AX,DATA   ;initialize DS
        MOV      DS,AX
        SUB      BP,BP     ;clear the
        SUB      SI,SI     ;addressing regs.
        SUB      DI,DI
        MOV      DL,[BP+SI] ;read memory with
        MOV      DH,[BP+DI] ;based indexed addr.
        RET

PROG    ENDP
CODE    ENDS
        END      CODE
```

Listing 3.
Laboratory Microsystems PC/Forth.

```
Screen # 75
  0 ( Lo Res Color Graphics                                        12/27/82 )
  1 FORTH DEFINITIONS HEX
  2
  3 ( initialize video controller chip )
  4 : TR              3D4 PC! 3D5 PC! ;
  5 : CRT_LRC_INIT  9 3D8 PC! 07F 4 TR 64 6 TR 70 7 TR 1 9 TR ;
  6
  7 ( initialize video memory map for low res graphics )
  8 : CLEAR_BUFF     4000 0 DO 0DE B800 I !L  2 +LOOP ;
  9
 10 ( set operating mode for low resolution color graphics )
 11 : LRCG           2 MODE  CRT_LRC_INIT  CLEAR_BUFF ;
 12 -->
 13
 14
 15


Screen # 76
  0 ( Lo Res Color Graphics                                        12/27/82 )
  1
  2 ( plot point in low res graphics:  x y color --- )
  3 : !DOT            ROT DUP 1 AND
  4                   IF    ROT 0A0 * + B800 SWAP 2DUP C@L
  5                         0F0 AND >R ROT R> OR ROT ROT C!L
  6                   ELSE  1 OR SWAP 10 * SWAP ROT 0A0 * +
  7                         B800 SWAP 2DUP C@L 0F AND
  8                         >R ROT R> OR ROT ROT C!L
  9                   THEN ;
 10
 11 DECIMAL
 12 -->
 13
 14
 15


Screen # 77
  0 ( Lo Res Color Graphics demonstration                           12/27/82 )
  1 3 VARIABLE COLOR
  2 0 VARIABLE CSEQ      2 , 4 , 6 , 8 , 10 , 12 , 14 ,
  3                   1 , 3 , 5 , 7 , 9 , 11 , 13 , 15 ,
  4 : SQUARE          DUP 80 + OVER 80 SWAP -
  5                   3 PICK DUP 50 + 50 ROT - SWAP 1+ SWAP
  6                   DO    DUP I COLOR @ !DOT OVER I COLOR @ !DOT
  7                   LOOP  2DROP DUP 50 + OVER 50 SWAP -
  8                   3 PICK DUP 80 + 80 ROT - SWAP 1+ SWAP
  9                   DO    DUP I SWAP COLOR @ !DOT
 10                         OVER I SWAP COLOR @ !DOT
 11                   LOOP  2DROP DROP ;
 12 : DEMO            LRCG 50 0
 13                   DO    I 15 AND 2* CSEQ + @ COLOR ! I SQUARE
 14                   LOOP  KEY DROP 3 MODE ;
 15
```

# HIGH QUALITY — LOW COST

OVERBEEK ENTERPRISES is committed to providing quality software at extremely low prices. We intend to make a handsome profit by having thousands of satisfied customers. We believe that the following two products represent truly outstanding bargains.

## $25

**Micro-WYL**
A powerful, Z80 CP/M text editor

Tired of trying to use ED under CP/M? This is the editor developed by Realworld Software, Inc. and reviewed in Infoworld (11/15/82).

*– Here are quotes from customers and reveiwers –*

"Micro-WYL is undoubtedly the hottest software bargain on the market"

"thank you, thank you, thank you"

"Micro-WYL is truly terrific"

*– Those are quotes from unsolicited letters from our customers.*

---

"While I am an avid Wordstar user, I personally prefer the ease and convenience of Micro-WYL when writing programs. The price is right and the product is great. Try it. This editor has numerous features not described in this review – all of which help to make this product an essential part of your program base."

*– From a review in the PASCAL MT Users Group Newsletter.*

---

"This editor is perfect for writing in nearly any programming language. [I] . . . find myself looking for excuses to use Micro-WYL, and certainly have no hesitation in recommending it to anyone whose requirements match the capabilities of this inventive piece of software."

*– From a review in Infoworld (11/15/82)*

---

We do offer a great editor at an unbelievably low price. WYLBUR[2] has been popular on IBM mainframes for over a decade.

Now you can have the convenience of WYLBUR on your micro.

*[1] CP/M is a registered trademark of Digital Research, Inc.*

*[2] WYLBUR is a registered trademark of The Board of Trustees of the Leland Stanford Junior University*

## $29.95

**DISK INSPECTOR**
a program that runs under Z80 CP/M[1] for disk inspection and modification

Have you ever been unable to read a file due to a bad sector? Have you ever erased the wrong file? Disk Inspector acts as a full-screen editor for diskettes. You can simply watch as sectors are displayed on the screen in both character and hex formats. When you wish to make the display pause, touch the spacebar. If you wish to alter a sector, it is a simple matter to move the cursor over the appropriate character, alter it, and have the sector rewritten.

Although Disk Inspector runs under CP/M you can inspect and alter normal (non CP/M) Apple diskettes, as well. The disk drives may be single or double density, single or double sided. The comprehensive manual will show you how to:

*Recover an erased file.*
*Modify a director entry.*
*Clean up a directory.*
*Utilize the CP/M Auto-Load feature.*
*Create multiple directory entries.*
*Read and modify non CP/M diskettes.*

The Disk Inspector is a full-screen editor for disks. Our competitors offer products in the $100–$200 range. We certainly invite comparison of this product with any comparable system in terms of features or user friendliness. In terms of price, there is no comparison.

*Note: Disk Inspector requires an 80 x 24 screen on your CRT and is currently available only in 8" SSSD, Kaypro, Apple/Softcard, NEC, and Altos Series 5 formats.*

☐ 8" SSSD
☐ Apple/Softcard
☐ KAYPRO II 5"
☐ NEC 5"
☐ ALTOS Series 5
☐ Televideo TS-802
☐ Osborne
☐ Superbrain
Northstar 5" DD
☐ Advantage
☐ Horizon

Amount: $25 for Micro-WYL _____
$29.95 for Disk Inspector _____
$2 for postage & handling _____
**Total** _____

Name _____
Address _____
City _____ State _____ Zip _____

Make your check out to:

**Overbeek Enterprises**
P.O. Box 726
Elgin, IL 60120

Circle **no. 18** on reader service card.

# AUGUSTA, Part II

## The Augusta P-Code Interpreter

by Edward Mitchell

Part 1 of this four-part series described the Augusta language. This month's article continues with a description of the p-codes and the p-code interpreter that executes Augusta programs.

### Stack Machines

The Augusta compiler translates source statements into code for a hypothetical "pseudo-machine" based on the use of stacks. Stacks provide a natural representation for arithmetic expressions and for implementing nested procedure calls and local data definitions.

Owners of Reverse Polish Notation (RPN) calculators are already familiar with stacks and their use for evaluating expressions. The basic operations performed on the stack are push and pop, to add and to remove data to and from the stack, respectively.

A stack is often compared to a spring operated stack of plates in a cafeteria. A new plate is added by *pushing* it on to the top. When a plate is removed, the whole stack *pops* upwards. For example, if we push both 13 and 10, the stack looks like

```
10
13
```

Then, we can add them together to give

```
23
```

If we translated these steps to an assembly-like language, we might write

```
PUSH 13
PUSH 10
ADD
```

Complicated expressions like 5*(3+7) are coded as

```
PUSH 5
PUSH 3
PUSH 7
ADD       ; Add 3+7
MULT      ; Multiply result by 5
```

With a few more instructions, the stack becomes a general purpose expression evaluator. For example, to perform the assignment, X:=10+13 on a stack machine, the following sequence of instructions might be executed

```
PUSH address of X
                ; Put the memory address
                ; of X on stack
PUSH 10
PUSH 13
ADD             ; Compute 10+12,
                ; leaving 23 on stack
STORE           ; Store top of stack value
                ; into the address con-
                ; tained in the next word
                ; on the stack
```

The Augusta p-machine consists of about 80 instructions (see Table 1) to manipulate the stack and perform other functions such as conditional testing, branching, procedure calls and returns.

### P-Machine Overview

The p-machine is a 16-bit stack-oriented hypothetical computer. It has a code pointer register CP that contains the byte address of the next instruction to execute, and a stack pointer SP that points to the top of the stack. In addition, the global frame pointer GF and the local frame pointer LF are used to speed up references to global and local variables, respectively.

Registers SB (Stack Base) and CB (Code Base) are of use when debugging programs and point to the bottom of the stack, and to the first byte of code for the currently executing procedure. CS (Code Segment) points to the address of the first byte of code. The number of the procedure that is executing is stored in the PN register (the first "PROCEDURE" seen in a program becomes procedure 1, the second "PROCEDURE" statement becomes procedure 2, and so on).

Before a program can be run, it must be loaded into memory. One more register, PT, contains the address of the procedure table. The procedure table contains a 7-byte entry for each procedure in the program and includes (1) the address in memory of the procedure's code, (2) the

### Corrections to Augusta, Part I

Several last minute editing changes were made by both the author and the editors at *Dr. Dobb's* in order to shorten the length of Part 1. Unfortunately, a few incorrect and misleading statements crept into the text.

The Augusta compiler is written in Microsoft BASIC, and hence, can be run on systems other than a standard CP/M system. However, compiled programs can only be executed by using the p-code interpreter, a program that currently runs only on Z-80-based CP/M systems. While I do hope to produce an MS-DOS version, it was premature for the editor's note to announce that such a version will be available shortly.

Originally, Part 2 of the series described a BASIC language implementation of the p-code interpreter. But that description was removed and some of the comments in Part 1 no longer apply. For example, the BASIC p-code interpreter implemented virtual memory but the assembly-coded version of the interpreter does not.

The variable declaration

```
S : STRING;
```

defines S as a string of up to 80 bytes (not 128, as shown in Part 1). A different length is specified by writing

```
S : STRING(200);
```

which gives S a length of 200 bytes. Maximum string length is 254 bytes.

Part 1, Table 1 summarized only a few of the available "built-in" procedures and functions. With nearly 40 procedures, Augusta provides an Ada-like file system, printer and serial port I/O, string searching, MOVELEFT, MOVERIGHT, and a very limited dynamic memory allocation scheme.

To support file access, a fifth predefined type is used, as

```
F : IN_OUT_FILE;
```

which defines F as a file access variable. File variables may be grouped in arrays and passed as OUT procedure parameters.

The syntax diagrams contained three minor errors. Corrections are shown in Figure 1 (opposite). Procedures and functions are not required to have parameter lists as was implied by the original diagrams, and an ELSE may not follow an ELSEIF.

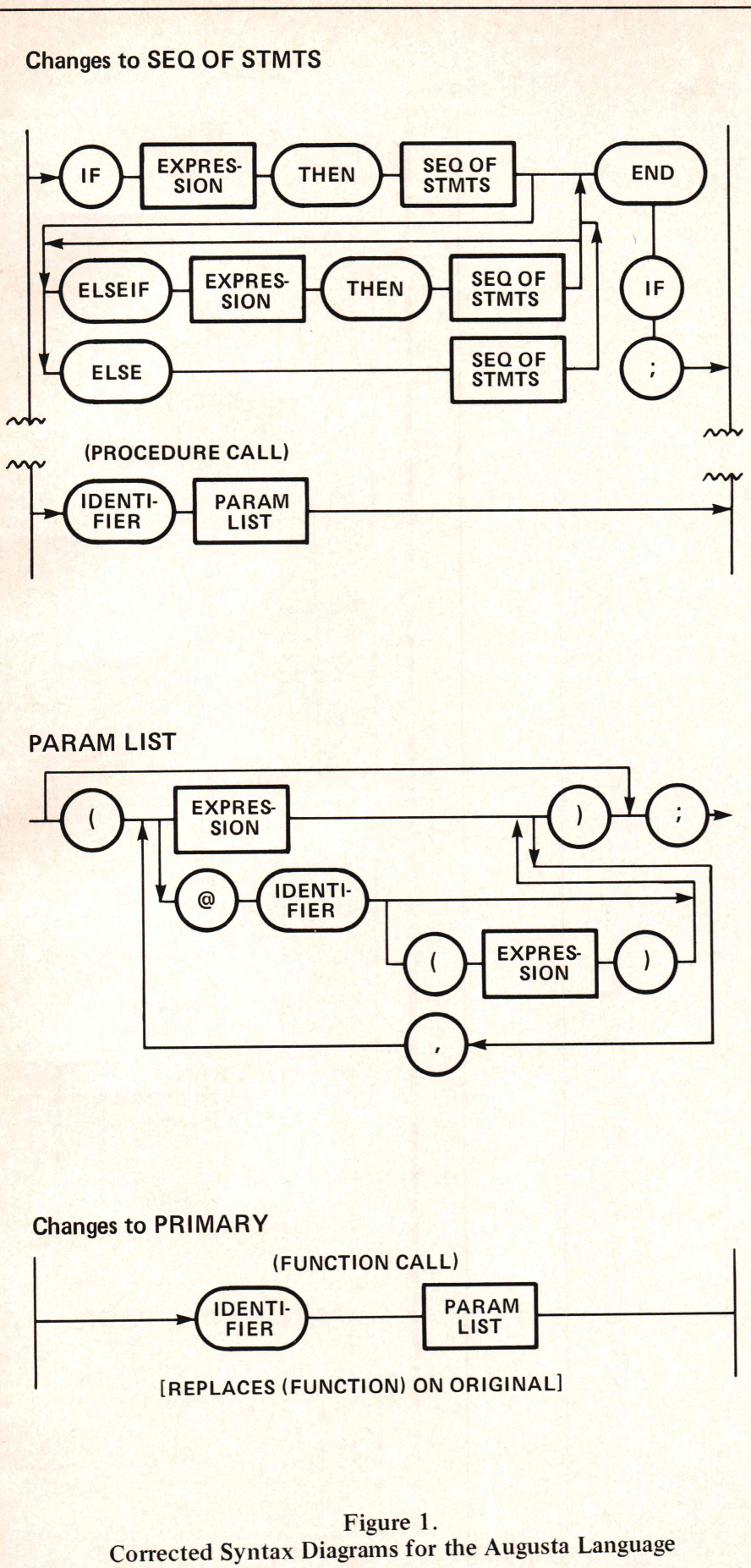



**Figure 1.**
**Corrected Syntax Diagrams for the Augusta Language**

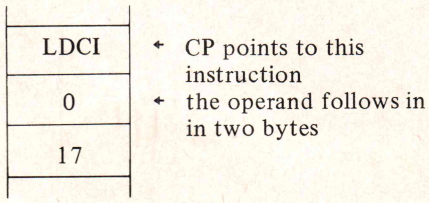
number of bytes of parameter variables, (3) the amount of memory required for local variables, and (4) the "lexical level" of the procedure.

All procedure calls are made through the procedure table. The instruction CGP 5, for example, means to call procedure 5. The location of procedure 5 and how much memory to allocate for its local variables are fetched from entry 5 in the procedure table.

## The P-Code Instructions

The entire group of instructions is divided into several smaller groups consisting of (1) "load" and "store" instructions, (2) string assignments, (3) logical operations, (4) integer arithmetic operations, (5) integer comparisons, (6) string comparisons, (7) jump instructions, and (8) procedure calls (see Table 1, pages 31-33).

A simple operation is to load a constant on to the stack. The LDCI or "Load Constant Integer" opcode is a single byte long, followed by a one-word (16-bit) value. When the p-code interpreter is executing instructions, CP always points to the next byte of the program code. For LDCI, the p-machine reads the instruction and decodes the opcode byte. Then it jumps to the appropriate LDCI subroutine to load the constant. The actual constant is stored in the two bytes following the instruction, as shown here:




After the LDCI opcode is read, CP is incremented to point to the first byte of the constant. The LDCI subroutine then reads the 16-bit integer and pushes it on the stack. A push is performed by writing the word into the memory location pointed to by SP, and then incrementing SP by two (since one word occupies two bytes and SP always points to a word). CP is incremented so that it points to just past the second byte of the constant, which is the next instruction.

Four data types are recognized by the p-machine: (1) 8-bit bytes, (2) 16-bit integers, (3) fixed-length strings, and (4) boolean values. An integer takes two bytes and represents numbers in the range -32768 to 32767. A string uses a fixed amount of memory determined from its declaration in the Augusta program, plus one byte to hold the current length of the string. For example, if S is a string of up to 80 characters in length, then 81 bytes

of storage are reserved. The first byte holds the length as illustrated here:

| Len | | | ... | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | | 79 | 80 |

For example, the string "Alphabet" would be stored in memory as

| 8 | A | l | p | h | a | b | e | t | ... |
|---|---|---|---|---|---|---|---|---|---|

A boolean TRUE is any non-zero word while FALSE is a word containing zero.

A variety of instructions are used to load the values or addresses of both global and local variables. Several variations on these instructions are provided to access "intermediate level" variables (these occur when procedures are defined inside of procedures that are defined inside of procedures, ad infinitum) and special "short" forms that reduce the size of the compiled programs.

A single instruction "STO" writes the word on the top of the stack into the address contained in the second word from the top of the stack.

Strings are never loaded on to the stack. Instead, just a pointer to the string is loaded. For example, when the statement

```
S := "STRING";
```

is compiled, Augusta translates this to a "Load Constant Address" (LCA) instruction, followed by the string assignment operator SAS. The constant string "STRING" is translated by Augusta into

```
LCA 6 STRING
      ↑
Length Byte
```

When the p-machine encounters the LCA opcode, it pushes the address of the length byte on to the stack. Then it sets CP to point to the first byte after "STRING."

Logical and arithmetic instructions perform an operation to the top one or two words of the stack. For example, ADI pops the top two words from the stack, adds them together and then pushes the result. Relational operators, like EQUI, pop the top words and compare the values. If the relation is true, then a non-zero word is pushed, otherwise a zero is placed on the stack.

## Stacks and Procedure Calls

When a program executes a procedure call, the computer uses a stack to keep track of the "return address." Each time that a procedure is called, the return address is pushed. If a called procedure, in turn, calls another procedure, then both return addresses are distinct. The return address of the most recently called procedure is always the topmost address on the stack.

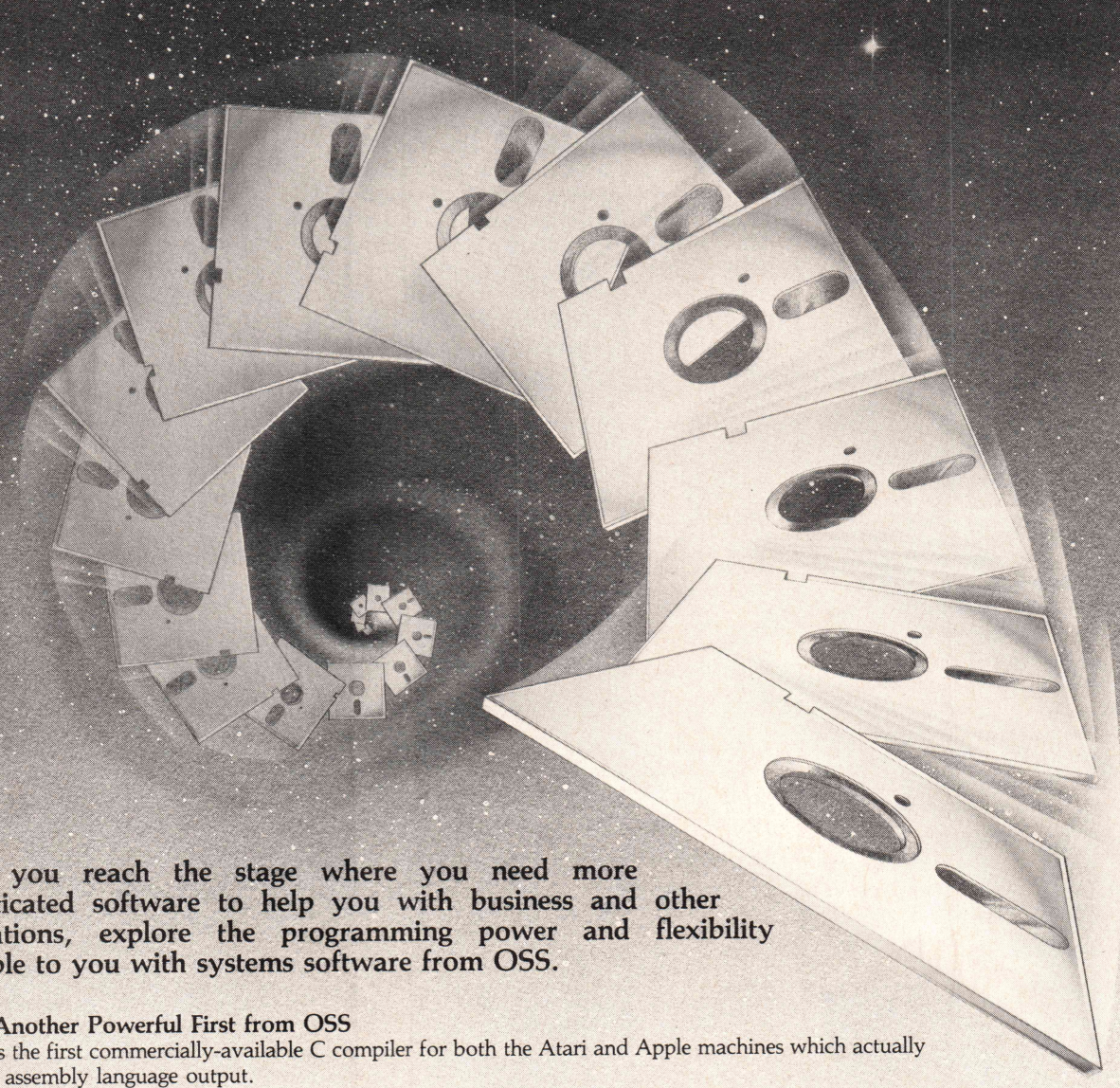
When a procedure finishes, it pops its return address from the stack and resumes execution at its calling point. As we'll see in a moment, the stack is also used to hold additional information. The portion of the stack that holds the return address is known as the "frame mark." The frame mark delineates each procedure from the other procedures on the stack and saves the state of the procedure that was running before the procedure call.

## Variable Allocations

Augusta permits *nested* procedure definitions. Each procedure can have its own *local* data. Space for local variables is allocated when the procedure is called. When the procedure exits, all of the space used by the local variables is made free.

Figure 2 (page 24) shows how a stack machine allocates storage space for variables defined in the outermost procedure and for local variables defined within called procedures. The variables in the outermost procedure are accessible to all of the procedures that are contained within it and are referred to as "global"

# EXPLORE A NEW DIMENSION IN SOFTWARE



**When you reach the stage where you need more sophisticated software to help you with business and other applications, explore the programming power and flexibility available to you with systems software from OSS.**

### C/65—Another Powerful First from OSS

C/65 is the first commercially-available C compiler for both the Atari and Apple machines which actually produces assembly language output.

C/65 supports a very usable subset of the extremely powerful and popular C language. Just as C is used by the most sophisticated programmers from the professional and academic communities, so shall C/65 prove to be a powerful and much-needed tool for 6502 software developers.

C/65 supports INTegers and CHARacters, arrays thereof and pointers thereto. Naturally, it also features full recursion, easy assembler interface, #INCLUDE, and a non-macro version of #DEFINE. AUTOmatic, global and EXTERNal variables are also available. When used with our MAC/65 assembler, C/65 is a powerful and flexible tool...$80.00

### A Strong Software Family

Other major systems software products from OSS include:

| | |
|---|---|
| BASIC A+ | the only logical upgrade to Atari BASIC with extra features for games and business programs....$80.00 |
| MAC/65 | the finest and fastest complete 6502 macro assembler/editor package you can buy....$80.00 |
| TINY C | for structured programming, an easy-to-use interpreter, a learning tool....$99.95 |
| BUG/65 | a powerful, self-relocatable debugger. FREE with MAC/65....$34.95 |

### And More...

OS/A+, the first and finest operating system for BOTH Atari and Apple II computers, is **NOW** included **FREE** as a part of every OSS systems software package. OS/A+ features a keyboard-driven, easy-to-use command processor, several simple resident commands, and logical and readable requests for even the most sophisticated utility commands. Versions of OS/A+ for some higher capacity drives are available at extra cost.

NOTE: Unless otherwise noted, all **OSS** products require 48K and at least one disk drive.

**ASK YOUR DEALER,** or call or write for our brochure.



**OSS** **Optimized Systems Software, Inc.** 10379 Lansdale Avenue • Cupertino • California • 95014 • (408) 446-3099

variables. Global storage space appears on the bottom of the stack.

When a procedure is called, any needed parameters are pushed, a frame mark containing the return address and other information is output, and the stack pointer is incremented high enough to leave space above the frame mark for local variables. When the procedure finishes executing, the information from the frame mark is restored, and the stack pointer is cut back to where it was before the parameters were pushed. In this manner, all local storage space is allocated dynamically at the time of the procedure call, and then disappears upon return to the caller.

Global variables are referenced as an offset from the global frame mark at the base of the stack. For example, a variable at offset 2 is referenced with the "Load Global Value" instruction

```
LDO 2     ; Load Global instruction
```

The location of the global frame is kept in register GF. To fetch the value at offset 2, the p-machine adds 2 to GF which computes the memory address of the global memory word.

Local variables are referenced as an offset from the local frame mark so that the instruction

```
LDL 3     ; Load Local instruction
```

references the data three bytes above the local frame mark. Register LF always contains the address of the currently executing frame.

Parameter variables are also referenced with the local data instructions. Since parameters are located *below* the frame mark, the local instruction uses a negative offset. For example,

```
LDL -16  ; Load the value of
         ; first parameter
```

## Handling Nested Procedures

Augusta, like Ada and Pascal, allows nested procedure definitions as in Figure 3 (page 28). Procedure P4, inside P3, can reference variables in both P3 and P2. These variables are neither "global" nor "local" to P4. Instead, they are "up-level" references.

After P4 has been called, the stack is as shown in Figure 3(b). To reference variable Y in procedure P3, the p-machine has to refer to the previous frame on the stack. Similarly, to reach variable X, the p-machine must traverse backwards two frame marks to reach the local data of procedure P2. Because of up-level references, the frame mark needs to contain more than just the return address.

The return address or "dynamic link" chains together the frame marks in the order that the procedures were called. If procedure P4 makes a recursive call on procedure P3, P4 *cannot* be the previous frame for an up-level variable reference. If it were, then P3 could up-level address variables for a lower level procedure. Therefore, the dynamic link is insufficient

**(Figure 2a)**

```
PROCEDURE P1 IS
  I : INTEGER;
  J : INTEGER;


PROCEDURE P2 ( X : INTEGER ) IS
  A : INTEGER;
  B : INTEGER;


BEGIN
  .
  .
  .
END;


BEGIN
  P2(3);
END;
```

**(Figure 2b)**

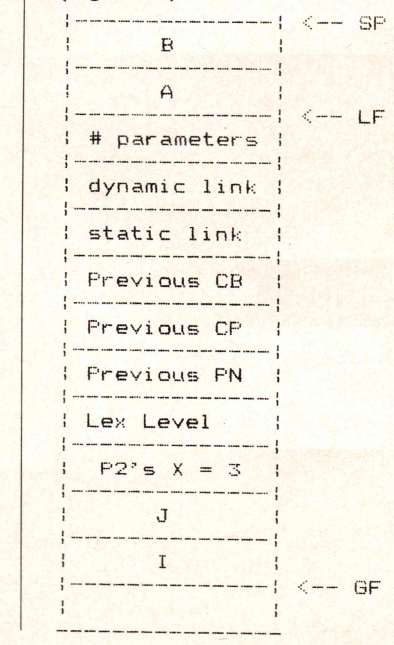
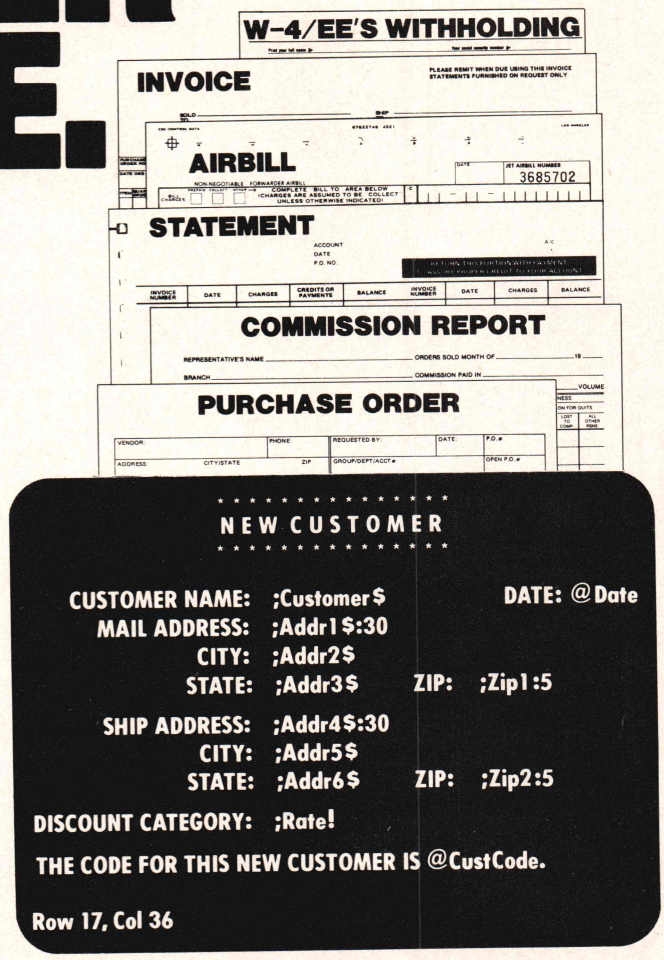



**Figure 2.**

**Storage for variables is allocated on the stack. (b) is a map of the stack for the program shown at (a), after P2 was called. Space for I and J is allocated at the base of the stack. When P2 is called, its single parameter, 3, is pushed on the stack and topped with the frame mark for P2 and P2's local variables.**

# END THE PAPER CHASE.




Now you can computerize your business forms and input screens without all that tedious, time consuming programming.

With ZIP,™ it's going to seem almost too easy.

## Just "paint" the screen.

To prepare an input screen or output form, just move the cursor around the screen and type in text, prompts and data field names exactly where you want them. Use "@" to print or display values, use ";" for input fields.

When the screen looks like the format you want, type "/S" and what you see is what you'll get. In seconds, ZIP writes all the program code to recreate the format on the screen or on paper.

And you can use the ZIP code immediately just by adding a single line to your main program: GOSUB xxxxx in BASIC, DO Formname in dBASE II.

## ZIP is quick and friendly.

ZIP runs on virtually every 8-bit micro known to man, and your terminal continues to work the way it did (tab, arrows, etc.), so you don't have to learn or unlearn anything about your equipment.

Commands are typed (no control codes), so you or your secretary can breeze through input screens and output forms up to 88 lines long and have ZIP whip out the BASIC or dBASE II code.

The ZIP Talker,™ a line at the bottom of the screen, always tells you exactly where you are. And Help is just two keystrokes away.

## Now MBASIC really ZIPs.

The MBASIC version goes further and gives your programs the same screen handling characteristics that ZIP has, by writing a piece of itself in MBASIC so that you can use it in any of your programs.

The operator can use the arrows, etc. during data entry and conveniently jump back and forth between the input fields.

You can specify field lengths, or let ZIP default to the available space. Either way, text and prompts are protected no matter what kind of terminal you have, so the operator can't write over the fields and prompts.

You get the MBASIC code for a "Talker" that you can use to pretty up your program prompts.

And easy, one-line data validation is built in.

## Join thousands of users ZIPping along.

All you need is an 8-bit micro with CP/M or MPM, 48k of memory and a 24x80 ASCII or ANSI terminal (Osborne 1 and 56k Apple okay, too).

The MBASIC and CBASIC versions are $160 each ($225 for both) plus $7 shipping (VISA, MasterCharge or money order). The dBASE II version is available alone from Ashton-Tate (213-204-5570), or we'll sell you dBASE II with ZIP for $650. For more information, contact Nexus, 5455 Wilshire, Suite 802, Los Angeles, CA 90036.

Or if you'd like to end the paper chase sooner, just call 800-227-3747. (In California, call 213-937-0554, add 6% tax.)

ZIP™

from **Nexus**

**The man-machine connection.**

for up-level addressing. Instead, a *static link* is used to reflect the structure of the program.

When P3 calls P4, the static link of P4's frame mark points to the frame mark of P3. If P4 calls P3, a new frame is created for P3 on top of the stack. However, P3's static link is set to point to the frame for P2, because P3 is declared inside of P2.

In addition to the dynamic and the static links, the frame mark also contains the number of bytes of parameters, the lexical nesting level, and the code base pointer of the current procedure. The lexical level is the depth of nested procedure definitions. If P2 is inside of P1, the first procedure in the program, then P2 is at lexical level 2. Similarly, if P3 is defined inside of P2, then P3 is at lexical level 3.

## Code File and Memory Layout

The p-code interpreter is contained in a code file named RUN. (Note: An alternate p-code interpreter named DRUN contains a built-in debug utility. The utility provides program tracing, single-step and *n*-step execution, and examining and changing the value of variables.) The interpreter reads the code file into memory and begins execution.

The code file is organized into three sections (see Figure 4(a), page 29): (1) a 128-byte header block containing information about the code file, (2) a 1792-byte procedure table containing space for 256 procedures (entry 256 is not used), and (3) the p-code for each of the procedures in the program.

When the code file is loaded, only the used portion of the procedure table is read into memory. Obviously, if the program only contains ten procedures, then only the first part of the table needs to be read.

After the code is loaded, the CP register is initialized to the first p-code in procedure 1, and the stack pointer is set to point to the first word after the p-codes. The other registers, including GF, LF, and PT, are set as needed. Figure 4(d) shows the memory layout after a code file has been read into memory.

## Augusta Efficiency

One way to measure a language's efficiency is to run a benchmark test and compare the size of the generated code and its execution time to that of other languages. Listing 1 (page 35) is the source code of a simple prime number generator based on the sieve of Eratosthenes. (Listing 2, also on page 35, is the disassembly of that same program's object code into p-codes.) Table 2 (page 34) compares that program, written in Augusta, to versions written for several other well-known compilers (see references 1 and 2, below).

## In the May Issue

The Augusta compiler is a one-pass recursive descent compiler. As the parser recognizes a language construct, it generates the appropriate code. Part 3 provides a general description of recursive-descent compilers.

(Figure 3 on page 28)
(Figure 4 on page 29)
(Figure 5 on page 30)
(Table 1 on page 31)
(Table 2 on page 34)
(Listings 1 and 2 on page 35)

DDJ

**References for Part 2.**

[1] Gilbreath, Jim. "A High-Level Language Benchmark," *BYTE*, September 1981, p. 180.

[2] Gilbreath, Jim and Gary Gilbreath. "Eratosthenes Revisited," *BYTE*, January 1983, p. 283.

# PERIPHERAL VISION

Floppy Disk Services, Inc. is a contracted SIEMENS drive dealer. Do not let the prices fool you, we buy in very large quantities to get the best price and pass that savings on to you! All systems are of the highest grade components and our cabinets are custom designed with you in mind! If not 100% satisfied, call us and we will promptly refund your money. †

**We carry Add-on drives for IBM, Radio Shack, Heath, Apple and most other microcomputers.**

| | |
|---|---|
| Apple II Add on drives | $329.00 |
| Apple 8 inch controller | 365.00 |
| Apple 80 track dual system | 1395.00 |
| Apple dual 8 inch system w/ controller | 1250.00 |
| FDD-100-5b 'flippy' exact HEATH add on | 235.00 |
| FDD-200-5 double sided 40 track drive | 250.00 |
| FDD-111-5 5ms step IBM or MOD 3 Add on | 245.00 |
| FDD-221-5 5ms step 80 track DD/DS | 350.00 |
| TEC SFD-51B 5ms 5¼ 48TPI | 215.00 |
| FDD-100-8d 8 inch single side DD drive | 340.00 |
| FDD-200-8p Double sided 8 inch drive | 445.00 |
| Custom 8 inch and 5¼ inch enclosures | Call |

System packages available for all drives ...

| | |
|---|---|
| Dual 8 inch system with EVERYTHING | 935.00* |
| Dual double sided 8 inch system | 1125.00* |
| Single 5¼ Heath or MOD I Add on w/ case | 285.00* |
| Dual 5¼ Heath or MOD I | 585.00* |
| 10mb Hard Disk for any computer | 2700.00* |
| CDR controller, allows any combo 8 and 5¼ inch drives to be added to your H88 or H89 | Call |

* 8 inch systems require minor assembly. Add $100.00 A&T. All 5¼ inch systems come assembled and tested.

† Equipment must be in same condition as you received it.

## WE HAVE ZENITH Z-100 SYSTEMS IN STOCK.

**Have a disk drive in need of repair? We have expert techs ready to optimize your drives! Call us for info.**


AT FLOPPY Disk Services, we not only sell drives, we also sell custom enclosures. Our cabinets are designed by our experts to be functional and attractive. And our quantity pricing is so attractive, we invite dealers and large group purchasers to call.

CALL TODAY ▪ 609-799-4440

IF you don't see what you want, give us a call between 9 am & 5 pm (ET). Chances are we'll have what you need for your system at your price. Due to production deadlines for advertising, prices in this ad are 2 months old, so we encourage you to call us for current prices and new product information.

PAYMENT POLICY – We accept Mastercard, VISA, personal checks & MO. We reserve the right to wait 10 working days for personal checks to clear your bank before we ship. All shipping standard UPS rates plus shipping & handling. NJ residents must add 5% sales tax.

PRICES & SPECIFICATIONS SUBJECT TO CHANGE

**FLOPPY DISK SERVICES INC.**

*741 ALEXANDER ROAD*
*PRINCETON, NJ 08540*

MOD I-II-III, CP/M are trademarks of Tandy and Digital Research respectively.

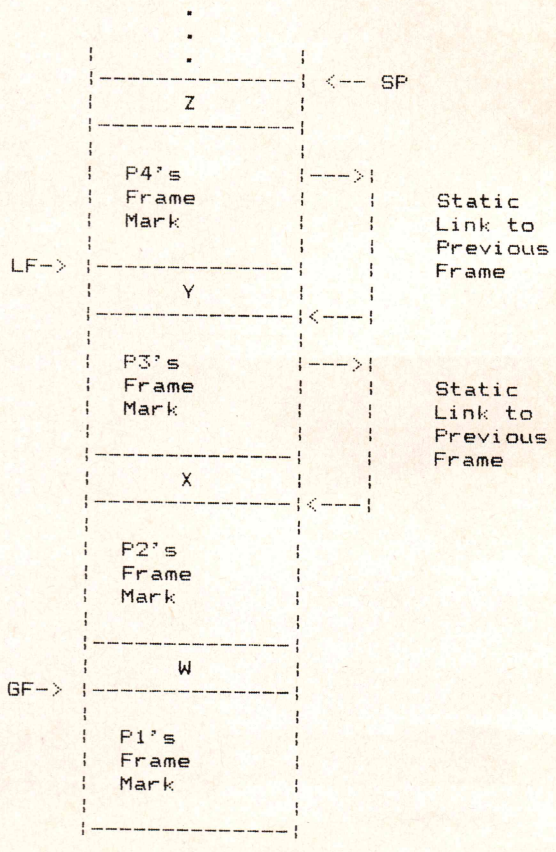
FD2-1

Circle **no. 25** on reader service card.

**(Figure 3a)**

```
PROCEDURE P1 IS
  W : INTEGER;

  PROCEDURE P2 IS
    X : INTEGER;

    PROCEDURE P3 IS
      Y : INTEGER;

      PROCEDURE P4 IS
        Z : INTEGER;
        BEGIN
          Z := Y;    -- ACCESS Y IN P3
          Z := X;    -- ACCESS X IN P2
        END; -- P4

      BEGIN
        Y:=1;
        P4;
      END; -- P3

    BEGIN
      X:=2;
      P3;
    END; -- P2

  BEGIN
    P2;
  END; -- P1
```

**(Figure 3b)**




## Figure 3.

**Procedures can be defined inside of other procedures, as shown at (a). When P4 references variable Y, the p-machine must traverse down the stack to the previous frame mark to reach Y. Similarly, a reference to X is a reference to two frames lower in the stack.**

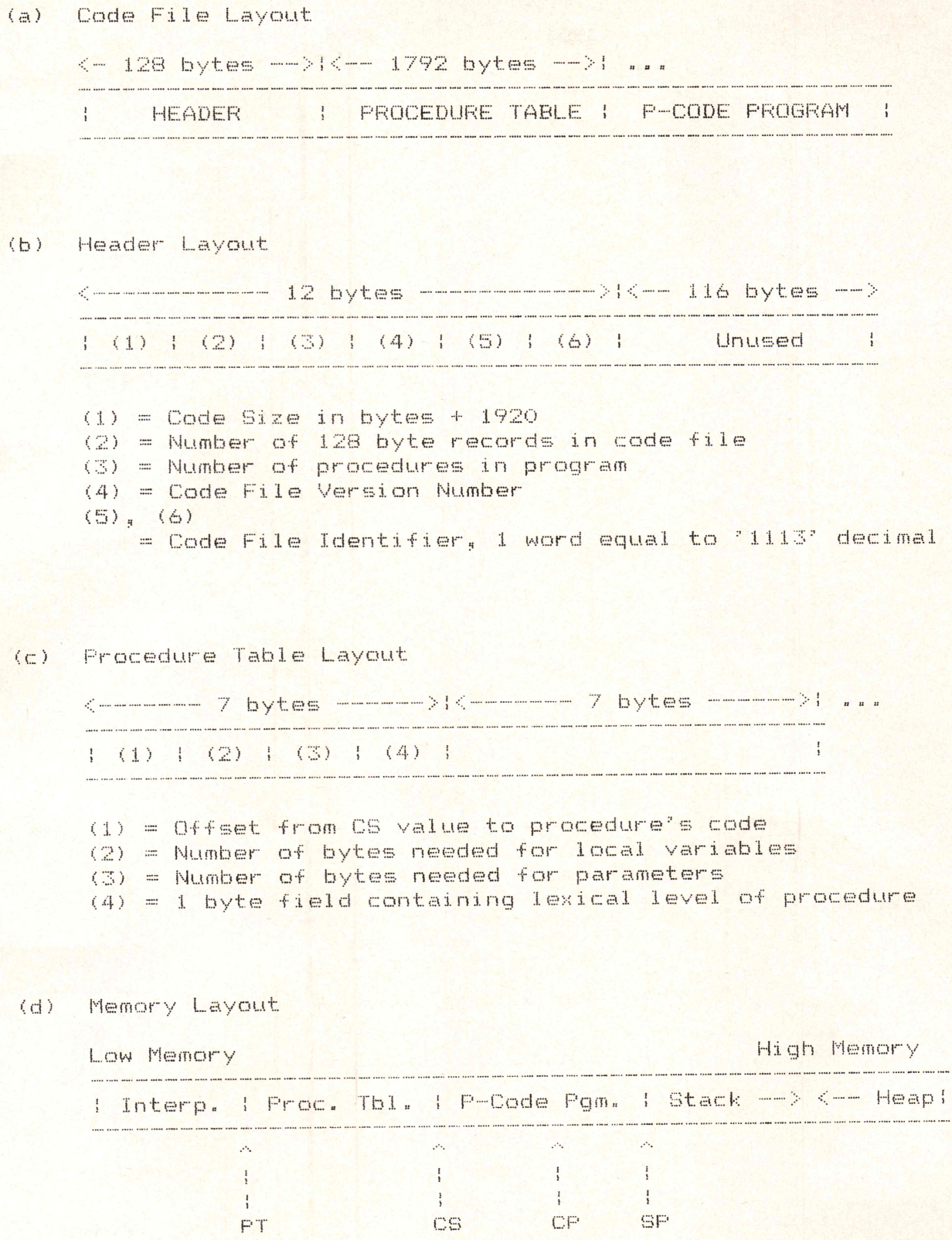
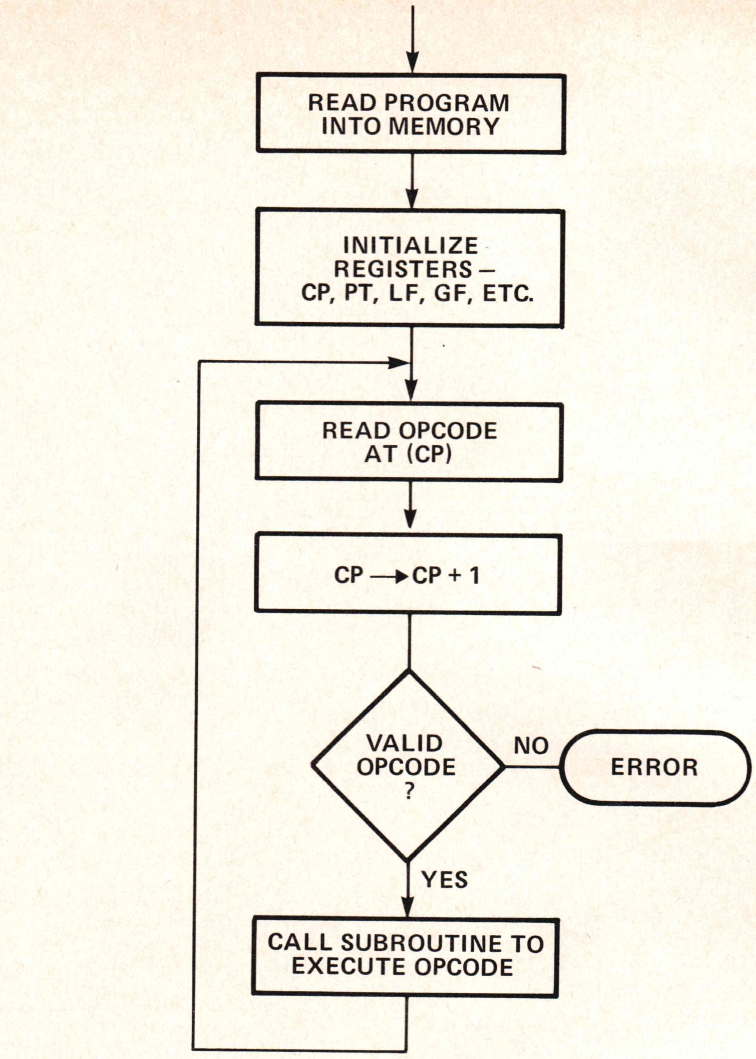



**Figure 4.**

**Detailed layout of an Augusta code file and of memory after a program has been loaded. (a) shows the code file format, (b) and (c) detail the format of the header block and of the proceedure table, and (d) shows the memory layout of a loaded program.**




**Example SUBROUTINES**

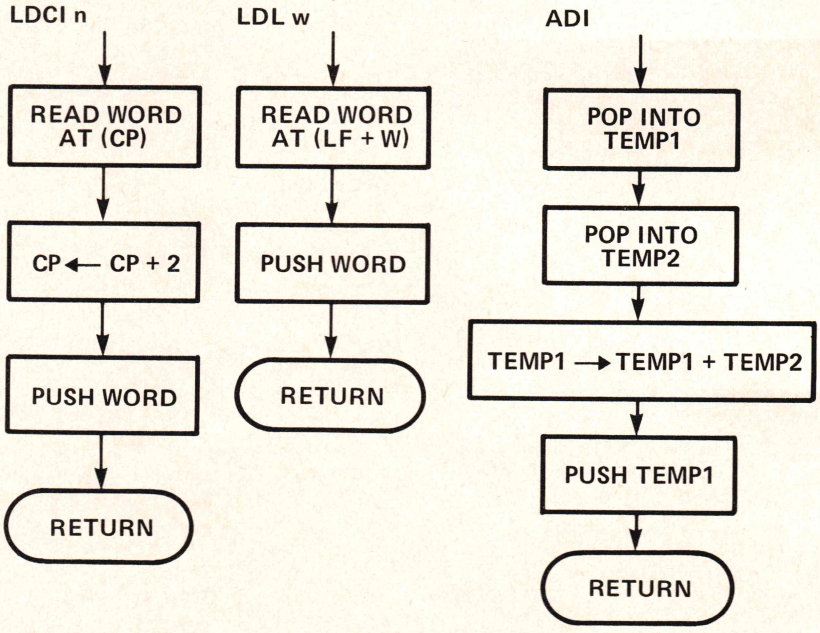



**Figure 5.**
**Outline of Interpreter Logic**

## Table 1.

### The Augusta P-Code Instruction Set.

**All of the Augusta p-code instructions are shown in Table 1. The number in parenthesis is the opcode value. A "b" following the opcode indicates that there is a one-byte parameter. "w," "w1," and "w2" are 16-bit words. In the descriptions, TOS is the word on the top of stack, and TOS-1 is the word just below the top of stack.**

Loads and Stores

```
( 1) LDCI w     - Load Constant Immediate Word

( 2) LDL w      - Load Local word at offset w

( 3) LLA w      - Load address of Local word at offset w

( 4) LDB        - Load the byte pointed to by the pointer in TOS.

( 5) LDO w      - Load global word at offset w

( 6) LAO w      - Load global address at offset w

( 8) LOD b, w   - Load word up level b static links at offset w

( 9) LDA b, w   - Load address up level, as above.

(11) STO        - Store indirect TOS into address in TOS-1

(12) SINDO      - Load indirect using the address in TOS
```

Short Loads

```
(57) SLDO b     - Short Load Global data at offset b.  b is in
                  the range 0..255.

(58) SLAO b     - Short Load address at global offset b.   b is in
                  the range 0..255.

(59) SLLA b     - Short Load Local Address at local offset b.

(60) SLDL b     - Short Load Local data at local offset b.

(61) SLDC b     - Short Load Constant in the range 0..255.

(49) to (56) SLDL0 .. SLDL7
                - Single byte opcodes to Short Load Local Data at
                  local offset 0 through 7.

(63) SLDCN1     - Short load negative 1 (-1)

(64) to (79) SLDC0 .. SLDC15
                - Single byte opcodes to short load a constant in
                  the range 0 to 15.
```

Array Indexing Instructions

```
(24) IND        - Index an integer array.  TOS-1 is the address
                  of the base of the array and TOS is an index
                  into the array.  IND pops TOS, multiplies it
                  by 2 and adds it to TOS-1, pushing the result.
```

*(Continued on page 32)*

**(Table 1 continued from page 31)**

```
(48) IXA b
                - Index array.  This is a generalized IND opcode,
                  where b substitutes for the "2" multiplier.

String Assignments
(13) LCA b, <Chars>
                - Load address of a string constant.  b is the
                  number of bytes in the string address.  <Chars>
                  are the characters in the constant.  Pushes the
                  address of the byte containing b and increments
                  CP to point to the next instruction following
                  <Chars>.

(14) SAS        - String Assignment.  TOS is a pointer to source
                  string  and TOS-1 points to a destination.  Copies
                  bytes from the source string to the destination string.

Logical Operators
(16) AND        - Boolean AND of TOS and TOS-1

(17) OR         - Boolean OR of TOS and TOS-1

(18) NOT        - Boolean complement.  TOS <-- NOT TOS

Integer Arithmetic Operations

(19) ADI        - Add the integers in TOS and TOS-1.

(20) NGI        - Let TOS equal -TOS.

(21) SBI        - Subtract integers.  TOS-1 minus TOS.

(22) MPI        - Multiply integers in TOS and TOS-1.

(23) DVI        - Divide integers, TOS-1 divided by TOS.

(45) MODI       - TOS-1 MOD TOS

(80) INCL w     - Increment word at local offset w.

(81) DECL w     - Decrement word at local offset w.

Integer Comparisons
                The comparions push -1 if the result is true, or 0
                if the comparison is false.

(25) EQUI       - Compare TOS to TOS-1 for equality. (=)

(26) NEQI       - TOS <> TOS-1

(27) LEQI       - TOS-1 <= TOS

(28) LESI       - TOS-1 <  TOS

(29) GEQI       - TOS-1 >= TOS
```

(30) GTRI - TOS-1 > TOS

String Comparisons

TOS and TOS-1 are pointers to the strings to compare. Pushes -1 if the comparison is true or 0 if the comparison is false.

(31) EQUSTR - TOS-1 = TOS

(32) NEQSTR - TOS-1 <> TOS

(33) LEQSTR - TOS-1 <= TOS

(34) LESSTR - TOS-1 < TOS

(35) GEQSTR - TOS-1 >= TOS

(36) GTRSTR - TOS-1 > TOS

Jump Instructions

All jumps are made relative to the location of the instruction.

(37) UJP w - Unconditional jump to w bytes away from current instruction.

(38) FJP w - False Jump. Jump if TOS is 0.

(39) XJP w1, w2, w3
- Case jump instruction. See text for description.

Procedure Calls and Returns

(40) CLP b - Call local procedure b.

(41) CGP b - Call global procedure b.

(42) CSP b - Call system procedure b. System procedures handle all input/output.

(46) CIP b - Call intermediate level procedure b.

(43) RET - Return from a procedure.

(46) RNP - Return from a function by saving the TOS value setting SP back before the parameters and frame mark for the current procedure and then restoring the TOS value.

Miscelleneous

(15) EOP - End of Procedure mark. This opcode is never executed but is output to mark the end of a procedure's code.

**(End Table 1)**

| Language | Compiled (bytes) | Compile/load (seconds) | Execute (seconds) |
|---|---|---|---|
| Pascal MT+ | 308 | 102 | 19 |
| Janus Ada | – | – | 27 |
| UCSD Pascal, Z-80 | 282 | 14 | 239 |
| UCSD Pascal, TRS-80 Mod II | 282 | 60 | 274 |
| Augusta, Z-80 Osborne I | 162 | 30 | 303 |
| JRT Pascal V2.0 | 224 | 34.5 | 383 |
| Supersoft Ada | – | – | 422 |
| Pascal/M | 301 | 50 | 450 |
| JRT Pascal | 232 | 65 | 470 |
| CBASIC2 | – | 26 | 484 |
| Tiny-C Compiler | – | 96 | 930 |

Table 2.
**Comparison of Augusta with other languages. Chart indicates compiled program size and compilation/load and execute times for the Eratosthenes Sieve prime number generator. See references 1 and 2 (page 26) for details.**

## Listing 1.

**Augusta Source Code for Prime Number Generation**

```
 2    0     0      0 -- Eratosthenes Sieve Prime Number Program
 3    0     0      0 -- in Augusta
 4    0     0      0
 5    0     0      0 PROCEDURE PRIME IS
 6    1     0    -14   PRIME,K,COUNT : INTEGER;
 7    1     0      6   FLAGS : ARRAY(8190) OF BOOLEAN;
 8    1     0  16388
 9    1     0  16388 BEGIN
10    1     0  16388   PUTLINE("10 ITERATIONS");
11    1    17  16388   FOR  ITER IN 1..10
12    1    26  16388   LOOP
13    1    30  16390     COUNT := 0;
14    1    34  16390     FLAGS(0) := TRUE;  -- Initialize Array
15    1    40  16390     MOVELEFT ( @FLAGS(0), @FLAGS(1), 16378);
16    1    53  16390     FOR  I IN 0..8190
17    1    64  16390     LOOP
18    1    68  16392       IF  FLAGS(I)  THEN
19    1    78  16392         PRIME := I + I + 3;
20    1    90  16392         K := I + PRIME;
21    1    99  16392         WHILE K <= 8190
22    1   104  16392         LOOP
23    1   108  16392                 FLAGS(K) := FALSE;
24    1   115  16392                 K := K + PRIME;
25    1   123  16392         END LOOP;
26    1   123  16392         COUNT := COUNT + 1;
27    1   133  16392 --      PUTSTR("PRIME #=");
28    1   133  16392 --      PUTINT(PRIME);
29    1   133  16392 --      NEWLINE;
30    1   133  16392       END IF;
31    1   133  16392     END LOOP;
32    1   133  16392   END LOOP;
33    1   139  16390   PUTINT(COUNT); PUTLINE(" PRIMES");
34    1   160  16388 END;
```

## Listing 2.

**Disassembly of the Object Code Into P-Codes**

```
Dissassembly of Augusta Program:  SIEVE.CPL
Number of Procedures =  1
-----------------------------------

Procedure #  1
Local Size =  16392
Parameter Size =  0
Lex Level =  1


 Offset      Opcode Value     Parameters
(     0) :     13  LCA     13 "10 ITERATIONS"
(    15) :     42  CSP     2
(    17) :      3  LLA     16388
(    20) :     65  SLDC1
(    21) :     11  STO
(    22) :      2  LDL     16388
(    25) :     74  SLDC10
(    26) :     27  LEQI
(    27) :     38  FJP     115  =>  145
(    30) :     58  SLA0    4
(    32) :     64  SLDC0
(    33) :     11  STO
(    34) :     58  SLA0    6
(    36) :     64  SLDC0
(    37) :     24  IND
(    38) :     63  SLDCN1
(    39) :     11  STO
(    40) :     58  SLA0    6
(    42) :     64  SLDC0
(    43) :     24  IND
(    44) :     58  SLA0    6
(    46) :     65  SLDC1
(    47) :     24  IND
(    48) :      1  LDCI    16378
(    51) :     42  CSP     12
(    53) :      3  LLA     16390
(    56) :     64  SLDC0
(    57) :     11  STO
(    58) :      2  LDL     16390
(    61) :      1  LDCI    8190
(    64) :     27  LEQI
(    65) :     38  FJP     71  =>  139
(    68) :     58  SLA0    6
(    70) :      5  LDO     16390
(    73) :     24  IND
(    74) :     12  SINDO
(    75) :     38  FJP     55  =>  133
(    78) :     58  SLA0    0
(    80) :      5  LDO     16390
(    83) :      5  LDO     16390
(    86) :     19  ADI
(    87) :     67  SLDC3
(    88) :     19  ADI
(    89) :     11  STO
(    90) :     58  SLA0    2
(    92) :      5  LDO     16390
(    95) :     57  SLDO    0
(    97) :     19  ADI
(    98) :     11  STO
(    99) :     57  SLDO    2
(   101) :      1  LDCI    8190
(   104) :     27  LEQI
(   105) :     38  FJP     18  =>  126
(   108) :     58  SLA0    6
(   110) :     57  SLDO    2
(   112) :     24  IND
(   113) :     64  SLDC0
(   114) :     11  STO
(   115) :     58  SLA0    2
(   117) :     57  SLDO    2
(   119) :     57  SLDO    0
(   121) :     19  ADI
(   122) :     11  STO
(   123) :     37  UJP     -27  =>  99
(   126) :     58  SLA0    4
(   128) :     57  SLDO    4
(   130) :     65  SLDC1
(   131) :     19  ADI
(   132) :     11  STO
(   133) :     80  INCL    16390
(   136) :     37  UJP     -81  =>  58
(   139) :     80  INCL    16388
(   142) :     37  UJP     -123  =>  22
(   145) :     57  SLDO    4
(   147) :     42  CSP     4
(   149) :     13  LCA     7 " PRIMES"
(   158) :     42  CSP     2
(   160) :     43  RET
```

**End Listing**

# A Small-C Operating System

by Brian McKeon

Want to experiment with operating systems? Well, here is the source code of a system written in the programming language "C" and developed using Ron Cain's Small-C compiler (*DDJ* No. 45).

As I was lucky enough to have had access to a UNIX system, I was able to bring up the Small-C compiler with a minimum of fuss. After a couple of weeks of spare time, I was able to return to the microprocessor system and run the compiler. It was then that the contrast between the large UNIX system and a micro system became obvious. In particular, I missed the ability to redirect input/output via the operating system. With UNIX, any program output destined for the console can be redirected to go to a file. Console input to a program can also be arranged to come from a file. Both functions can be easily specified on the command line (see "echo" later).

Many of the operating systems for micros were developed in the days when 32Kbytes of memory was a large amount for a microprocessor system. Now it is by no means unusual for even personal machines to have 64Kbytes. A portion of this extra capacity could be devoted to making the micro operating system more flexible. Initially, I was simply trying to add routines to the micro to implement I/O redirection.

It was an ideal opportunity to try out the Small-C compiler. However, after a very short time I found that I seemed to have a great many of the routines of a full operating system. As an exercise, I then decided to have a go and write a full system and this article is the result.

I have incorporated the UNIX process of having directories as files themselves. While this may seem unusual, it does lead to advantages in that many of the system routines, available for file access, can be used to process directory information. User programs can just as easily access directories as files.

The system loads and executes user programs at 100 hex to make possible the execution of CP/M-compatible programs if the source code is not available. These programs would require some form of interface to translate system calls from CP/M convention to the form expected by the described system. As an alternative, the system calls described could be altered to give a CP/M-compatible interface. While it is possible to add such an interface, this may not be worth the effort. The book, *Software Tools*[1], gives the source code of many useful programs, even a sophisticated editor. These are written in Ratfor but it is not a difficult task to translate to "C" and this would allow the use of more sophisticated system calls and make possible the transfer of programs to other systems.

In all of my work so far, I have used the Small-C compiler as published (*DDJ* Nos. 45, 48) with modifications enabling backlash escape sequences in literal characters (*DDJ* No. 56) and the removal of the bugs in the arithmetic rotate routines (*DDJ* No. 52). One of the more advanced versions now being sold would make the translation and compilation of any published "C" programs easier. Remember that source code will be required if you want to bring up the compiler on a new system.

Characteristics of this operating system are: input/output redirection on the command line; hierarchical directories (directories of directories and files); directories able to be processed as files; a copy in memory of the pointers used in file maintenance (giving fast file access); a disk group of 512 bytes (could be altered); or pointer range of 32767 disk groups; parsed command line arguments passed to user programs (see UNIX argc, argv[ ]); and dynamic memory allocation of 512-byte blocks.

A disadvantage of the system as it stands is that it has been developed for a one-disk-drive machine. Simultaneous access of multiple drives would require either swapping of the directory pointer blocks to/from the drives or maintenance of all pointer arrays in memory. While using more memory, the second alternative is the more time efficient. Allocating the top 16K of memory to the system program and the pointer arrays would allow the addressing of greater than 1Mbyte of disk space. The upper limit of disk space is the pointer range of 32767 disk groups or 16 megabytes with 512 byte group size. Large system size would make memory storage of arrays wasteful and an alternative file structure is described towards the end of this article that would be a good basis for larger capacity systems.

## File Structure

There are three pointer arrays being swapped between disk and memory: glinks[ ], loptr[ ], and hiptr[ ] (see Listing on page 41). The largest array is glinks[ ]. For any group "i" on disk, the next group in the linked list from group "i" would be glinks[i]. If glinks[i] is zero, it indicates the end of a list of groups, i.e., end of a file (Figure 1, page 37). There is a linked list for each file, a list of free groups left on disk, and even a short linked list that the system uses to load and save an image of the arrays glinks-loptr-hiptr on disk.

Files are tied into a directory structure by the arrays loptr[ ] and hiptr[ ]. A directory is simply a linked list of files and this is achieved using the array loptr[ ] with loptr[k] used to index the next file in loptr[ ]. See Figure 2, page 37, for a pictorial representation of this and the following arrangements. If loptr[k] is zero, then the last file in the current directory has been reached. If the value of hiptr[ ] for a given directory element is positive, it is taken to be a pointer to the first group of a file, with the array glinks[ ] linking further groups of the file. If negative, this element of the directory is taken to be a further directory and the negative value is complemented and used to index back into the directory array loptr[ ].

The first entry in a directory is always a file of the *names* of files in that directory, one name per line. As an end-of-line character is used as a name delimiter, there is virtually no limit to name length. The *n*th line refers to the *n*th entry along the loptr[ ] list for that directory. If there are 32 lines in the file of filenames for a directory, then it must be possible to index through the loptr[ ] array 32 times and find a zero in that element of loptr[ ], i.e. the expected end of that directory list of files.

The first file name in a directory file is always a file called "." and is a reference to the file of filenames for that directory.

*Brian McKeon, C.N.P.U., Ward 4, Prince Henry Hospital, Little Bay, NSW, Sydney, Australia.*

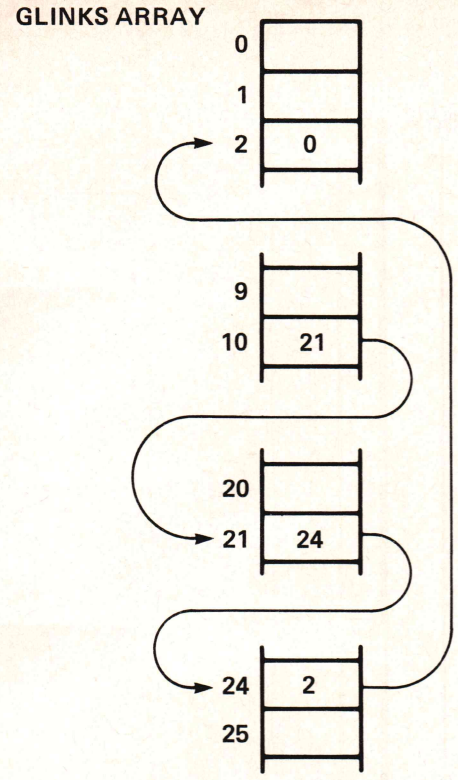



Figure 1.

Usage of array glinks[ ] to link groups of a file with first group number 10. The file consists of groups 10, 21, 24, and 2, in that order.

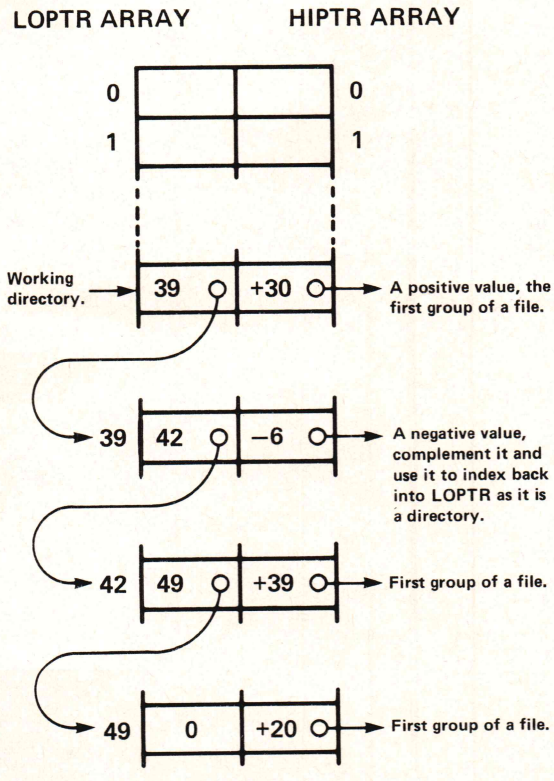



Figure 2.

An example of a directory consisting of four entries, the second of which is another directory, the others files.

On power-up, the system will always get loptr[2] (one of a few reserved locations) and assume that this pointer points to the user's directory. When the system opens a file, given the name, it scans the directory (see "scndir(name)," system call) and when the line number of the entry is known, it indexes into the loptr[ ] array this number of times. If the hiptr of this entry is negative, the entry must be another directory; if positive, it is the first disk group of a file entry.

As already mentioned, redirection of console input and output has been incorporated in this sytem. When the character ">" is encountered on the command line, the next characters are assumed to be a filename, a file of that name is created, and any program output that would have been put to the console will go to that file. Similarly, the character "<" indicates that any requests by the executed program for console input will be taken from the file whose name follows the "<" character.

## System Overview

The flexibility of the overall system becomes obvious when the simple program "echo" is considered:

```
main()
    {while(putchar(getchar()));}
```

Execution of this program echoes typed characters. They will be doubly echoed, as I have written getchar to echo any non-control characters. If, on the command line, we say "echo >newfile," any characters typed at the console will be redirected into the file "newfile." If a file by that name already exists, then it will be replaced. To terminate the process, type control d. This gives the end of file code. Then, on the command line "echo<newfile"; the file will then be listed to the console, as the input to echo has been redirected from console to come from "newfile." "echo <newfile >newerfile" gives a copy of newfile into newerfile. Now "echo <.", the contents of the file ".", which is the file of filenames for the current working directory, will be shown.

When the user changes his working directory using the system call "chdir," the value of loptr[ ] for the new directory is stored in loptr[3]. The default directory for the user is in loptr[2]. If "chdir" is called with a zero argument, the user is returned to his default directory from wherever he is. If a program to be executed cannot be found in the working directory, the system directory will be searched by default. The system directory is pointed to by loptr[1].

A list of the system calls is in Table 1. The listing of "library for Small-C system programs" gives details of how getchar and putchar are formed. A function that illustrates some system calls and is also required by the Small-C compiler on the new system is that of "fopen" (note that I have altered the second argument to be an "int").

```
fopen(name,func)
    char name[ ];
    int func;
    {
    int buffa;
    if((buffa=alloc())==0)return 0; /* get a buffer */
    if(func==rdfn)return (openf(name,buffa,rdfn));
    if(func==wrfn)return (creatf(name,buffa));
    dalloc(buffa); /* error-give buffer back */
    return 0;
    }
```

## System Transfer

It is not enough to have a copy of the system code on the disk, as the initial directory also needs to be there and, as there are no intrinsic functions in the system, there should be at least a memory to disk save program on the disk as well. My memory save program takes a numeric argument and saves that number of groups beginning at a predetermined location. One way to get going would be to put sectors onto the disk manually but there is another easier way. I started debugging the system by assembling it down low in memory and having the disk I/O routines simply doing block moves to and from higher memory. Simple programs and text could be loaded into memory at the appropriate location and loaded by the Small-C system as if coming from disk.

At this stage, the directory loader routine generated a dummy directory with three files, each of four groups (2K), a free group list and a free pair (loptr-hiptr) list. The free group list is simply a long file pointed to by hiptr[2]. The free pair list is a linked list of any spare loptr[ ] cells and is used when a new file or directory is created. This linked list of cells is pointed to by hiptr[1]. When the bugs were out of the system, I altered the disk write routine to actually write information to the disk. I then had a suitably formatted disk for the new system. After copying this disk (twice), I recompiled the system to reside in high memory, altered the disk read to come from disk instead of memory, and the system was up.

The system code takes up about 9.5K and the pointer buffers about 2.5K for my 250K drive. One system call I will be changing in the future is "scndir." Filenames will be allowed to contain "?" (match with any character) and "*" (match with any number of characters). A second parameter will also be passed and this will tell scndir what line/entry in the directory to begin the search with. Another area that I will

| System calls | Descriptions |
|---|---|
| puterr(str) | Put a string to the console (cannot be redirected, used for error messages). |
| getc(unit) | Get a character from "unit," unit=0 standard input, unit=1 the communication port, returns character or 0 on end of file. |
| putc(c,unit) | Put a character to "unit" (see getc), returns "c". |
| readf(unit) | Read the next sequential group on opened file, returns 0 on end of file. |
| writef(unit) | Write the next sequential group. |
| openf(name,buffer,fn) | Open "name" for read(fn=1) write(fn=2) or append (fn=3) with "buffer" as the buffer address, returns a unit number or 0 if not found. |
| create(name,buffer) | As for openf, write assumed, any old file by that name is lost. |
| closf(unit) | Close the given unit number, flushes write buffer if required. |
| readr(unit,N) | Read *n*th group from opened unit number, returns 0 if out of file group range. |
| writer(unit,N) | Write *n*th group to opened unit number, returns 0 if out of range. |
| scndir(name) | Scan the working directory for "name," returns number of entry in directory, 1 to *n* or 0 if not there. |
| chdir(name) | Look for "name" in the current directory and if it is a directory, change operations to it, if name==0 changes back to root directory of user. |
| mkdir(name) | Create a new directory "name," returns a 0 if name in use. |
| rmfil(name,i) | Remove directory entry "name," if not found, returns 0. Won't allow rmfil(".". If "i" is 0, won't allow directory entry removal. If "i" is 1, a recursive remove of the entry is undertaken, directory or file. |
| alloc() | Return the address of the next free top 512 byte block of memory. Addr will lie on a 512 byte boundary. |
| dalloc(addr) | Give back a 512 byte block of memory for future use. |
| dirfn(i) | For directory information: i=0 returns ptr to glinks[], i=1 returns ptr to loptr[], i=2, returns ptr to hiptr. |
| dirio(rw) | Refresh or save the directory pointer arrays: rw=1 (write) gives a save to disk, rw=2 (read) gives a refresh from disk. |

**Table 1.**
**System Calls.**

be modifying will be the system of group linking for disk files. The group links will be part of the files on disk eliminating the large array "glinks" (Figure 3).

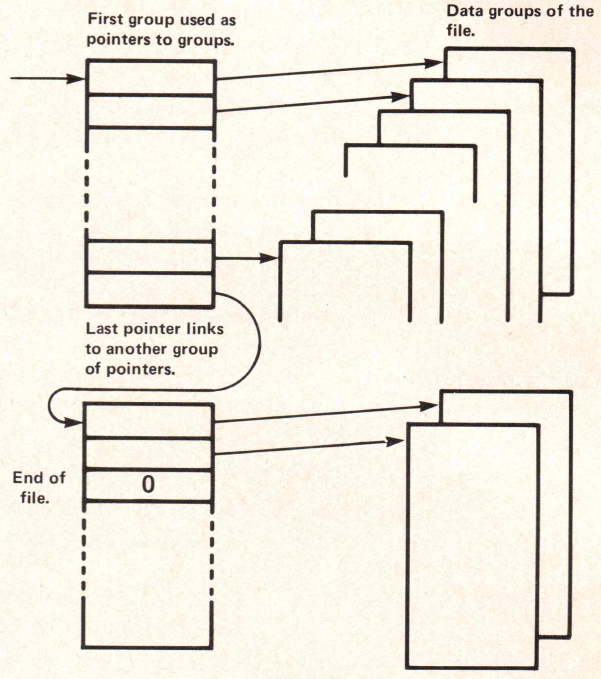



Figure 3.
**A proposed file structure that would remove the need for an in-memory array of group links (glinks).**

## Conclusion

While I have a compiler on the system (courtesy of Ron Cain), I do not yet have an assembler for it. If any reader has written an assembler (in Small-C?) for 8080 mnemonics, I would like to hear from them. Good luck!

**(Listing begins at right)**

DDJ

*Editor's Note: For those who have not yet implemented the Small-C compiler published in* DDJ, *some back issues are still available. The greatly improved Small-C v.2, by J.E. Hendrix, is available in issues #74 and #75. Ron Cain's version one of the compiler, along with bug fixes and patches, can most easily be found in our Bound Volumes.*

## Reference

[1] Kernighan, B.W. and Plauger. *Software Tools*. Addison-Wesley, Reading, Massachusetts, 1976.

# Small-C Operating System (Text begins on page 36)

## Listing One

```
/** operating system written in Small-C     **/
/** Brian McKeon 5/5/82                     **/
#define eol 10          /* end of line char */
#define bufsiz     512  /* disk buffer size */
/** some memory control                     **/
#define usrprg 256 /* start and entry point of user program */
int memmap[8];          /* bitmap of 512 byte blocks of memory */
/** and directory structure                 **/
#define dirone 22  /* ptr to 1st grp of glinks-loptr-hiptr */
#define lnksiz 500 /* one link per 'bufsiz' bytes */
int glinks[lnksiz];
#define ptrsiz 50  /* seems to be enough */
int loptr[ptrsiz];
int hiptr[ptrsiz];
/** First few pointer pairs have special functions **/
/** zero ptrs are set to to 0 as traps              **/
#define rsvptrs 4  /* reserved system pointer pairs */
#define sysdir loptr[1] /* ptr to system dir struct */
#define userdir loptr[2] /* ptr to root dir of user */
#define workdir loptr[3] /* ptr to -current- user dir structure */
#define freptr hiptr[1] /* ptr to free dotted pair list */
#define grpfree hiptr[2] /* ptr to first free disk group */
/** i/o structures          **/
int stdout;         /* and for standard output */
#define maxunits 10     /* max no. simul open files */
#define minunit 2  /* 0, 1 reserved for console and communication port */
int grw[maxunits]; /* 0 if free, 'rdfn' for read and 'wrfn', write */
#define rdfn  1
#define wrfn  2
#define apfn  3    /* append, used for openf call only */
/** structures to keep track of file i/o **/
int firgrp[maxunits];   /* first group of a file group list */
int lnkgrp[maxunits];   /* link group for sequential i/o */
int gbuffs[maxunits];   /* addr of the buffers assoc with a unit */
int gptrs[maxunits];    /* pointers into these buffers */
char sysinp[bufsiz];    /* system input buffer */
char sysout[bufsiz];    /* and output buffer */
/** command line argument passing to user programs **/
int argc;           /* argument count */
int argv[10];       /* array of ptrs to argument strings */
/** line input buffer - argv[] strings **/
#define maxlin 128 /* max line input size */
char linbuf[maxlin];    /* line input buffer */
/** main entry point for system **/
/* call the system with the desired i/o parameters */
/* with arg0 the number of the system call  */
/*                              */
#asm
    org 0a000h
#endasm
            /* 0a000 gives blocks 1 to 79 spare */
#define maxblk 77  /* leave a couple of blocks for stack */
system(arg0,arg1,arg2,arg3)
    int arg0,arg1,arg2,arg3;
```

*(Continued on next page)*

```
{
int munit,savedir;
if(arg0==0)return (puts(arg1));     /* put str to std err */
if(arg0==1)return (getc(arg1));     /* char fetch */
if(arg0==2)return (putc(arg1,arg2)); /* char put */
if(arg0==3)return (readf(arg1));    /* sequential read */
if(arg0==4)return (writef(arg1));   /* sequential write */
if(arg0==5)return (openf(arg1,arg2,arg3)); /* open a file */
if(arg0==6)return (creatf(arg1,arg2)); /* create a file */
if(arg0==7)return (closf(arg1));    /* close a unit */
if(arg0==8)return (readr(arg1,arg2)); /* read random grp */
if(arg0==9)return (writer(arg1,arg2)); /* write random */
if(arg0==10)return (scndir(arg1)); /* scan dir for file name */
if(arg0==11)return (chdir(arg1));   /* change directory */
if(arg0==12)return (mkdir(arg1));   /* make new dir */
if(arg0==13)return (rmfil(arg1,arg2));  /* remove file-directory */
if(arg0==14)return (alloc()); /* get addr of free blk */
if(arg0==15)return (dalloc(arg1)); /* de-alloc a mem blk */
if(arg0==16)return (dirfn(arg1));   /* give addr of dir in mem */
if(arg0==17)dirio(wrfn); /* go system, save the dir */
else {inits();dirio(rdfn);
     puts("illegal system call >17: directory restored\n");
     }
while(1)  /* command interpreter loop */
     {
     inits();       /* re-initialize stk, i/o, hardware */
     puts("$ ");          /* prompt to user */
     if((munit=getlin(linbuf))==0)continue; /* get command */
     if(parslin(linbuf,munit)==0)  /* parse into argc, argv */
          continue;       /* on parse error */
     if((munit=openf(argv[0],sysinp,rdfn))==0)
          {
          if(workdir==sysdir){puts("open error\n");continue;}
          else
               {
               savedir=workdir; /* save the working dir */
               workdir=sysdir; /* and look in the sys dir */
               munit=openf(argv[0],sysinp,rdfn);
               workdir=savedir;
               if(munit==0){puts("open error\n");continue;}
               }
          }
     if(loadf(munit)==0)continue; /* and load the file */
     closf(munit);        /* close the loader unit */
     usrprg(argc,argv);   /* call the loaded user program */
     closf(stdout);       /* flush the output if any */
     dirio(wrfn);         /* and save the dir */
     }
}

/* get the next line of input from the console */
/* all white space is replaced by nulls (0) for parsing */
#define backsp 8
getlin(buffa)
    char buffa[];
```

```
    {
    int k;
    char c;
    k=0;
    while(k<maxlin)
        {
        c=inchar();
        if(c==eol)break;
        if(c==backsp)
            {
            if(k){--k;outchar(c);}
            continue;
            }
        if(c<=' ')c=0;
        buffa[k++]=c;
        }
    buffa[k]=0;
    return k;
    }
/* parse a line into argc, argv and also re-direct i/o */
parslin(str,nchar)
    char str[];
    int nchar;
    {
    int pt,endarg;
    char lastc;
```

*(Continued on next page)*

```
    pt=argc=0;
    argv[0]=&str[0];
    while(1)
        {
        while(str[pt])++pt; /* move over argument */
        while(str[pt]==0)++pt;   /* and nulls */
        if(pt>nchar)break;  /* finished? */
        if(str[pt]=='>')    /* check for re-direction */
            {
            if((stdout=creatf(&str[pt+1],sysout))==0)
                {puts("i/o redirection error\n");return 0;}
            }
        else if(str[pt]=='<')
            {
            if((stdin=openf(&str[pt+1],sysinp,rdfn))==0)
                {puts("i/o redirection error\n");return 0;}
            }
        else argv[++argc]=&str[pt];
        }
    return ++argc;
    }

/* open a given file name for file i/o */
/* from the current directory          */
/* rw is 'rdfn' for input, 'wrfn' for output and 'apfn' for append */
openf(name,buffer,rw)
    char name[],buffer[];
    int rw;
    {
    int filno;
    if((filno=scndir(name))==0)return 0;  /* find file name */
    return (openit(filno,buffer,rw));
    }

/* lower level open operates from number entry of file in dir */
/* for example 1 would open the dir file itself               */
/* the append function should only be used with null terminated */
/* files as it searches for this null char.      */
openit(filno,buffer,rw)
    int filno;
    char buffer[];
    int rw;
    {
    int i,k,grp;
    i=workdir;
    while(--filno)i=loptr[i]; /* down the dir list */
    if(hiptr[i]<0)return 0;    /* attempting to open dir? */
    if((filno=findio())==0)return 0; /* any i/o units left? */
    firgrp[filno]=hiptr[i];  /* pointer to first grp of file */
    gbuffs[filno]=buffer; /* setup buffer location */
    if(rw==rdfn)        /* read ? */
        {
        grw[filno]=rdfn;
        lnkgrp[filno]=hiptr[i];/* read from this first group */
        gptrs[filno]=bufsiz;     /* initialize the pointer */
        }
```

```
else if(rw==wrfn)          /* write ? */
     {
     frefil(hiptr[i]);     /* free up the old groups */
     hiptr[i]=0;           /* initially append null */
     grw[filno]=wrfn;      /* write */
     lnkgrp[filno]=-i;     /* by -, indicate a link to the dir! */
     gptrs[filno]=0;            /* initialize the pointer */
     }
else if(rw==apfn)          /* append ? */
     {

     /* get to the last group */
     grp=hiptr[i];
     if(glinks[grp]==0)lnkgrp[filno]=-i;
     else
          {
          while(glinks[glinks[grp]]!=0)grp=glinks[grp];
          lnkgrp[filno]=grp; /* 2nd last grp */
          grp=glinks[grp];   /* last grp */
          }
     if(grupio(grp,buffer,rdfn)==0)return 0; /* read last grp */
     frefil(grp);       /* and give it up */
     grw[filno]=wrfn;
     k=0;
```

*(Continued on next page)*

```
            while(buffer[k]!=0)++k;  /* go to the old eof */
            gptrs[filno]=k;      /* pointer now ready for append */
            }
        return filno;
        }
/* create a file */
creatf(name,buffa)
    char name[],buffa[];
    {
    int unit,k,l,newfree;
    if((grpfree==0)|(freptr==0)|(name[0]==0))return 0;
    if(unit=openf(name,buffa,wrfn))return unit; /* in the dir? */
    if((unit=openit(1,buffa,apfn))==0)return 0; /* no, open the dir */
    k=workdir;           /* and append the file */
    while(loptr[k])k=loptr[k];    /* go down the dir list */
    newfree=loptr[freptr];   /* get a new free pointer */
    loptr[k]=freptr;     /* and add to the directory */
    loptr[freptr]=0;     /* append a null */
    hiptr[freptr]=0;     /* and no groups */
    freptr=newfree;           /* up-date free pointer */
    l=0;      /* errors after this point are fatal */
    while(name[l]) /* append the name */
         if(putc(name[l++],unit)==0)   /* to the dir file */
             createrr(k);
    if(putc(eol,unit)==0)
         createrr(k);
    closf(unit);   /* and close the directory */
    return (openf(name,buffa,wrfn)); /* openf should now work */
    }

/* fatal error routine from create function */
/* dont flush the dir buffer as there is an error somewhere */
createrr(k)
    int k;
    {
    int oldfree;
    oldfree=freptr;
    freptr=loptr[k];
    loptr[k]=0;    /* re-terminate the dir */
    loptr[freptr]=oldfree;
    puts("fatal failure on file create\n");
    system(17,0,0,0);   /* and try to recover */
    }

/* close a given unit number */
closf(unit)
    int unit;
    {
    int k;
    k=1; /* default return value */
    if(unit<minunit)return 0;     /* trap illegal close */
    if((grw[unit]==wrfn)&(gptrs[unit]!=0))  /* write required? */
         k=writef(unit);
    grw[unit]=0;   /* now free up this unit */
    return k;
    }
```

```
/* look for a spare unit number else return 0 */
findio()
    {
    int k;
    k=minunit;
    while(k<maxunits)
         {if(grw[k]==0)return k;++k;}
    return 0;
    }
/* scan directory for a matching line to the given one */
/* and return a number 1 to N, else zero if not found */
scndir(name)
    char name[];
    {
    int linnum,munit,k,flg;
    char c;
    if(name[0]==0)return 0;
    if((munit=openit(1,sysinp,rdfn))==0)return 0; /* open dir for read */
    linnum=0;
    while(1)
         {
         flg=1;
         k=0;
         ++linnum;
         while(1)
              {
              if((c=getc(munit))==0){closf(munit);return 0;}
              if(c==eol)break;
              if(c!=name[k++])flg=0;
              }
         if((name[k]==0)&flg){closf(munit);return linnum;}
         }
    }

/* put an error message to the console */
puts(str)
    char str[];
    {
    int j;
    j=0;
    while((outchar(str[j++]))!=0);
    }

/* get a character from a given unit number */
getc(unit)
    int unit;
    {
    char c;
    if(unit==0)unit=stdin;   /* standard in? - re-direct */
    if(unit==0)return (inchar()); /* console port */
    if(unit==1)return (incom());  /* communication port */
    if(grw[unit]!=rdfn)return 0;  /* open for read? */
    if(gptrs[unit]==bufsiz)       /* get pointer - empty? */
         if(readf(unit)==0)return 0;
    c=*(gbuffs[unit]+gptrs[unit]);
    ++gptrs[unit];
    return c;
    }
```

*(Continued on next page)*

```
/* sequential read of file */
readf(unit)
    int unit;
    {
    int nxtgrp;
    if(grw[unit]!=rdfn)return 0;
    gptrs[unit]=0;
    nxtgrp=lnkgrp[unit];
    lnkgrp[unit]=glinks[nxtgrp];  /* update grp no. */
    return (grupio(nxtgrp,gbuffs[unit],rdfn)); /* refill buffer */
    }
/* random read of file group, range 0 to N */
readr(unit,grpno)
    int unit,grpno;
    {
    int rdgrp;
    if(grw[unit]!=rdfn)return 0;
    rdgrp=firgrp[unit];
    while(grpno--)rdgrp=glinks[rdgrp]; /* down the file grp list */
    return (grupio(rdgrp,gbuffs[unit],rdfn));
    }
/* put a character to a given i/o unit */
putc(c,unit)
    char c;
    int unit;
    {
    if(unit==0)unit=stdout;        /* standard out? - redirect */
    if(unit==0)return (outchar(c)); /* console port */
    if(unit==1)return (outcom(c)); /* communication port */
    if(grw[unit]!=wrfn)return 0;  /* write unit? */
    if(gptrs[unit]==bufsiz)        /* full buffer? */
        if(writef(unit)==0)return 0;
    *(gbuffs[unit]+gptrs[unit])=c;
    ++gptrs[unit];
    return c;
    }
/* write sequential to given unit */
writef(unit)
    int unit;
    {
    int newgrp;
    if(grw[unit]!=wrfn)return 0;
    gptrs[unit]=0;
    newgrp=fregrp();    /* get a new group */
    if(lnkgrp[unit]>0)glinks[lnkgrp[unit]]=newgrp; /* link */
    else hiptr[-lnkgrp[unit]]=newgrp; /* special case: link to dir */
    lnkgrp[unit]=newgrp;     /* and ready for next call */
    return (grupio(newgrp,gbuffs[unit],wrfn));
    }
/* random write of file group, range 0 to N */
writer(unit,grpno)
    int unit,grpno;
    {
    int wrgrp;
    if(grw[unit]!=wrfn)return 0;
```

*(Continued on page 50)*

```
    wrgrp=firgrp[unit];
    while(grpno--)wrgrp=glinks[wrgrp]; /* down the file grp list */
    return (grupio(wrgrp,gbuffs[unit],wrfn));
    }
/* free up a list of groups */
frefil(grp)
    int grp;
    {
    int fgrp; /* a pointer to the first free group */
    if(grp==0)return 0; /* dont lose the free list */
    fgrp=grpfree;  /* a pointer to the first free group */
    grpfree=grp;   /* point free list at the file */
    while(glinks[grp])grp=glinks[grp]; /* go to end of file */
    glinks[grp]=fgrp;   /* and link on the old free list */
    }
/* get a free group */
/* null terminate */
fregrp()
    {
    int grp;
    grp=grpfree;   /* find the next free group */
    grpfree=glinks[grp];     /* update the free pointer */
    glinks[grp]=0; /* and null terminate the returned group link */
```

```
    return grp;
    }

/* load a program, checking for oversize and for jump at start */
loadf(unit)
    int unit;
    {
    int grp,addr,blk;
    char ujump;
    grp=lnkgrp[unit];    /* get the first group */
    addr=usrprg;         /* load into the user area */
    blk=addr>>9;         /* and allocate mem as you go */
    while(grp)
         {
         if(blk==maxblk){puts("program too large\n");return 0;}
         if(grupio(grp,addr,rdfn)==0){puts("load error\n");return 0;}
         addr=addr+bufsiz;
         fixblk(++blk); /* allocate the mem */
         grp=glinks[grp];
         }
    ujump=*usrprg; /* should be the jump byte if a program */
    if(ujump!=-61){puts("not a program\n");return 0;}
    return 1;
    }
```

*(Continued on next page)*

## Small-C Operating System (Listing continued, text begins on page 36)

```
/* save/restore the directory structure to/from disk */
dirio(rdwr)
    int rdwr;
    {
    int grp,addr;
    grp=dirone;
    addr=glinks;
    while(grp)
        {
        if(grupio(grp,addr,rdwr)==0)return 0;
        grp=glinks[grp];
        addr=addr+bufsiz;
        }
    }
/* change work directory */
chdir(name)
    char name[];
    {
    int pt,n;
    if(name==0){workdir=userdir;return;}
    if((n=sendir(name))==0)return 0;
    pt=workdir;
    while(--n)pt=loptr[pt];  /* down the dir list */
    if(hir r[pt]>=0)return 0; /* not a directory */
```

```
    workdir=-hiptr[pt]; /* it is a dir, change */
    }
/* make a new directory */
mkdir(name)
    char name[];
    {
    int pt,n,newfree;
    if(n=openf(name,sysinp,rdfn))
        {closf(n);return 0;}     /* name is in use ! */
    if((n=creatf(name,sysinp))==0)return 0; /* cannot create */
    closf(n); /* got the name entry, close it */
    pt=workdir;
    while(loptr[pt])pt=loptr[pt]; /* name must be last on dir-list */
    newfree=loptr[freptr];   /* get the new free pointer */
    hiptr[pt]=-freptr;  /* link on a cell (- to indic dir) */
    pt=freptr;
    freptr=newfree;          /* update the freptr */
    loptr[pt]=0;   /* empty dir */
    hiptr[pt]=fregrp(); /* and the directory file grp itself */
    return (grupio(hiptr[pt],".\n",wrfn)); /* dir contains '.' only */
    }              /* i.e. itself */
/* remove a directory-files structure */
rmfil(name,flg)
    char name[];
    int flg;
    {
    int n,i,preptr,pt,postptr;
    if((n=scndir(name))==0)return 0; /* in the dir? */
    i=n;
    --i;
    if(i==0)return 0;   /* dont allow rmfil(".",x) */
    preptr=workdir;          /* and go to the link before the entry */
    while(--i)preptr=loptr[preptr];
    pt=loptr[preptr];   /* the entry */
    postptr=loptr[pt];  /* and the one after */
    if((hiptr[pt]<0)&(flg!=1))return 0; /* remove dir? */
    if(rmname(n)==0)return 0; /* now remove the n th. name */
    loptr[preptr]=postptr;        /* link around */
    loptr[pt]=0;        /* and terminate to restrain 'remove' */
    remove(pt);
    return 1;
    }
/* remove the n th. name from the work directory */
rmname(n)
    int n;
    {
    int obuf,ibuf,ounit,iunit,l,wrtflg;
    char c;
    wrtflg=obuf=iunit=ounit=0;
    while(1)  /* but only used for one-pass flow control */
        {
        if((ibuf=alloc())==0)return 0;
        if((obuf=alloc())==0)break;
        if((iunit=openit(1,ibuf,rdfn))==0)break;
        if((ounit=openit(1,obuf,wrfn))==0)break;
        l=1;
        wrtflg=1;      /* writing flag */
        while(c=getc(iunit))
```

*(Continued on next page)*

```
                {
                if(n==1)wrtflg=0;   /* suppress this entry? */
                if(wrtflg)putc(c,ounit);
                if(c==eol){++l;wrtflg=1;}
                }
            putc(0,ounit);
            wrtflg=1;break;
            }
        closf(iunit);closf(ounit);dalloc(ibuf);dalloc(obuf);
        return wrtflg; /* will be 0 on an error */
        }
/* recursive remove of dir-file structure */
remove(pt)
        int pt;
        {
        int newfree;
        while(pt) /* down the dir list */
                {
                if(hiptr[pt]<0)remove((-hiptr[pt]));    /* if a dir */
                else frefil(hiptr[pt]);         /* or if a file */
                newfree=pt;     /* now add this loptr.hiptr to free list */
                pt=loptr[pt];  /* pt onto next in list */
                loptr[newfree]=freptr; /* link on the old free list */
                freptr=newfree;     /* and update the free pointer */
                }
        }
/* return the addr of the next free top blk of mem */
alloc()
        {
        int blk;
        blk=maxblk;
        while(blk)
                if(blkfree(--blk))break;
        fixblk(blk);   /* fix this block */
        return (blk<<9);    /* and return the addr */
        }
/* fix a block of mem for use, given the block no */
fixblk(blk)
        int blk;
        {
        int bits,words;
        words=blk>>4;
        bits=blk-(words<<4);
        bits=-(1+(1<<bits));     /* complement for mask */
        memmap[words]=bits&memmap[words];
        }
/* is the given block number free ?: returns 1 if true, else 0 */
blkfree(blk)
        int blk;
        {
        int bits,words;
        words=blk>>4;
        bits=blk-(words<<4);
        return ((1<<bits)&(memmap[words]));
        }
```

```
/* free up the given block corresponding to the given addr */
dalloc(addr)
    int addr;
    {
    int bits,words,blk;
    addr=32767&(addr>>1);     /* take care to mask */
    blk=addr>>8;              /* and then calc the blk no. */
    if(blk==0)return;         /* trap */
    words=blk>>4;
    bits=blk-(words<<4);
    memmap[words]=memmap[words]|(1<<bits);
    }
/* return the location of a directory structure */
dirfn(arg)
    int arg;
    {
    if(arg==0)return glinks;
    if(arg==1)return loptr;
    if(arg==2)return hiptr;
    }
inchar()
    {
    char c;
    c=conin();      /*the machine code char input */
    if(c>=' ')outchar(c);     /*if not a control char */
    if(c==13)return outchar(eol); /* carr-ret to eol */
```

*(Continued on next page)*

```
    if(c==4)return 0;    /* control-d ret null */
    return c;
    }
outchar(c)
    char c;
    {
    if(c==eol)conout(13);     /* insert a carriage return if eol */
    return (conout(c));
    }
#include shlibr.c
/* this library is simply the routines ccgchar-cmpbcde from DDJ 48 */
```

## Listing Two

```
/* initialize stack, i/o and hardware etc. */
inits()
    {
    int i;
    #asm
    mvi  a,0c3h     ;fix up the system call location at zero
    sta  0
    lxi  h,system
    shld 1
    lxi  h,system-4
    pop  b     ;the local variable 'i'
    pop  b     ;and the return addr
    sphl
    push b     ;the return addr
    push b     ;and 'i'
    #endasm
    stdin=stdout=0;      /* std in, out via console */
    i=8;       /* now book up all memory */
    while(i)memmap[--i]=0;   /* and free up user area */
    i=maxblk;
    while(--i)dalloc(i<<9);  /* leaves zero block not free */
    i=0;
    while(++i<maxunits)grw[i]=0;
    loptr[0]=hiptr[0]=glinks[0]=0;      /* trap errors */
    }
/* get/put a group to disk          */
/* to/from a given dma address      */
/* returns 1 if o.k., 0 if error  */
#define sectrk 26        /* 26 by 128 byte sectors per track */
                  /* conversions in the next section */
                  /* assume a 512 byte bufsiz !!!       */
grupio(group,addr,io)
    int group,addr,io;
    {
    int track,sector,nemad;
    if(group==0)return 0;
    track=(group<<1)/13;
    sector=(group<<2)-track*26; /* 0 to 25 range not 1 to 26! */
/***** insert routines to read/write 4 sequential sectors ******/
    }
```

The following routines are some programs to get the system going.
The first is 'save'.

```
/* Save a specified number of blocks of memory into a specified file */
/* The blocks of memory are assumed to start at 2560 (0A00H)         */
#define bufsiz 512
char buff[bufsiz];
main(argc,argv)
    int argc,argv[];
    {
    int unit,ngrp,grp,addr;
    if(argc!=3){puterr("\nWrong no. of args");return;}
    if((unit=creatf(argv[1],buff))==0)
        {puterr("\nCannot open file");return;}
    if((ngrp=todec(argv[2]))==0){puterr("\nHow many blocks?");return;}
    addr=2560;      /* first location saved */
    while(ngrp--)
        {
        move(addr,buff);
        if(writef(unit)==0){puterr("\nWrite error");return;}
        addr=addr+bufsiz;
        }
    closf(unit);
    puterr("\nSuccessful save");
    }
```

*(Continued on next page)*

```
/* move a buffer in memory */
move(from,to)
    char from[],to[];
    {
    int k;
    k=0;
    while(k<bufsiz)
         to[k]=from[k++];
    }
/* convert a string to an integer: return 0 on error */
/* routine straight from small-c compiler   */
todec(str)
    char str[];
    {
    int k,n,c;
    n=k=0;
    while(c=str[k++])
         {
         c=c-'0';
         if((c<0)|(c>9))return 0;
         n=10*n+c;
         }
    return n;
    }
#include syslib.c
End of program.

This program 'rm' enables the removal of files.
It is set up so as to be not able to remove directories.
#include syslib.c
main(argc,argv)
    int argc,argv[];
    {while(--argc)rmfil(argv[argc],0);}
End of program.

The next program is 'stats'.
/* Gives statistics of number of free dotted pairs and groups left */
/* If any arguments are given it will give the size of these files */
#include syslib.c
main(argc,argv)
    int argc,argv[];
    {
    int pt,n,i,glinks[],loptr[],hiptr[];
    glinks=dirfn(0);
    loptr=dirfn(1);
    hiptr=dirfn(2);
    i=1;
    while(i<argc)        /* for any given file names */
         {
         if((n=sendir(argv[i]))==0)continue;
         pt=loptr[3];    /* the working dir ptr */
         while(--n)pt=loptr[pt];  /* go to the files dotted pair */
         pt=hiptr[pt];  /* a pointer to the file */
         if(pt>0)
              {
```

```
                n=size(pt,glinks);
                if(n<0)puts(" corrupted file structure in ");
                else {outdec(n);puts(" groups in");}
                puts(argv[i++]);puts("\n");
                }
            }
        pt=hiptr[1];
        n=size(pt,loptr);   /* find the no. of free nodes */
        if(n<0)puts(" free node list is corrupted\n");
        else outdec(n);puts(" free nodes left\n");
        pt=hiptr[2];
        n=size(pt,glinks);  /* find the no of free grps left */
        if(n<0)puts("\n free group list is corrupted\n");
        else outdec(n);puts(" free groups left\n");
        }
size(pt,array)
        int pt,array[];
        {
        int n;
        n=0;
        while(pt)
             {
             pt=array[pt];
             if(++n>32000){n=-1;break;}
             }
```

*(Continued on next page)*

# Small-C Operating System

(Listing continued, text begins on page 36)

```
    return n;
    }
puts(str)
    char str[];
    {
    int k;
    k=0;
    while(putchar(str[k++]));
    }
outdec(num)
    int num;
    {
    int k,zs;
    char c;
    zs=0;
    k=10000;
    if(num<0){num=(-num);putchar('-');}
    while(k>=1)
        {
        c=num/k+'0';
        if((c!='0')|(k==1)|(zs))
            {zs=1;putchar(c);}
        num=num%k;
        k=k/10;
        }
    }

/* library for small-c system programs */
#asm
sys equ  0
#endasm

/* some definitions */
#define rdfn 1    /* file read function */
#define wrfn 2    /* write function */
#define apfn 3    /* and append */
#define bufsiz 512
#define eol 10
puterr(str)
    char str[];
    {sys(0,str,0,0);}
getchar()
    {sys(1,0,0,0);}
putchar(c)
    char c;
    {sys(2,c,0,0);}
getc(unit)
    int unit;
    {sys(1,unit,0,0);}
putc(c,unit)
    char c;
    int unit;
    {sys(2,c,unit,0);}
```

```
readf(unit)
    int unit;
    {sys(3,unit,0,0);}
writef(unit)
    int unit;
    {sys(4,unit,0,0);}
openf(name,buff,fn)
    char name[],buff[];
    int fn;
    {sys(5,name,buff,fn);}
creatf(name,buff)
    char name[],buff[];
    {sys(6,name,buff,0);}
closf(unit)
    int unit;
    {sys(7,unit,0,0);}
readr(unit,n)
    int unit,n;
    {sys(8,unit,n,0);}
writer(unit,n)
    int unit,n;
    {sys(9,unit,n,0);}
scndir(name)
    char name[];
    {sys(10,name,0,0);}
chdir(name)
    char name[];
    {sys(11,name,0,0);}
mkdir(name)
    char name[];
    {sys(12,name,0,0);}
rmfil(name,flg)
    char name[];
    int flg;
    {sys(13,name,flg,0);}
alloc()
    {sys(14,0,0,0);}
dalloc(addr)
    int addr;
    {sys(15,addr,0,0);}
dirfn(arg)
    int arg;
    {sys(16,arg,0,0);}
gosys()
    {sys(17,0,0,0);}
#asm
;fetch a single byte from the address in hl into hl
The routines are as in DDJ 48 but only the routines
'ccgchar' to 'cmpbcde'
#endasm
```

**End Listing**

# 6809 Threaded Code

## Parametrization and Transfer of Control

by H. T. Gordon

High-level languages of the Forth type – now proliferating like the rabbits that inspired creation of Fibonacci numbers – are usually not compiled to machine-language "object" code. Their more-or-less humanly intelligible source code gets translated by a text-interpreter program into an intermediate code ("intcode"), a compact sequence of memory addresses (that point to a block of machine code) and parameters. This gets executed by an inner interpreter program that transfers control successively to the machine-code blocks, "threading" them. An example of a Z80 inner interpreter is given on p. 29 of the book by Loeliger (1981), who comments (pps. 36-7) on the timing inefficiency of threading. The overhead timing loss of executing a "primitive" command (defined by a machine-code block) is 76 cycles, while a "secondary" command (defined by an intcode sequence, a *de facto* subroutine) costs 268 extra cycles for the transfer of control. This must be why the fastest version of Forth in the benchmark-comparison study of Gilbreath (1981) ran *six-fold slower* than the truly-compiled PL-I/80 or Whitesmith's C (all on a Z80 system).

The recent evolution of Forth-like HLLs, facilitated by newer CPUs with superlative addressing power, has altered the threading mechanism in order to minimize timing losses. In the Z8000 Forth system of Odette (*DDJ* No. 71) each primitive machine-code block is terminated by two machine-code instructions (4 bytes, 18 cycles) that cause a jump to the next primitive in the intcode sequence, so that a sequence of primitives "runs itself." Even if the next command is a secondary, transfer of control to its intcode sequence is expedited by prefacing that with a machine-code CALL COLON (4 bytes, 10 cycles, plus 35 cycles execution time for the COLON subroutine). Since the terminal return to the main intcode sequence by the SEMI command costs 25 cycles, the total timing penalty is 70 cycles. Relative to the Z80, this is a four-fold reduction in timing loss, at the cost of two more bytes added to each command.

*H. T. Gordon, College of Natural Resources, University of California, Berkeley, CA 94720.*

The purpose of this note is to demonstrate that the Motorola 6809 can "thread" intcode with even greater efficiency, since its instruction set is rich in both direct and indirect indexed autoincrement codes. If the design makes use of its Y-index register as the equivalent of a CPU program counter, terminating each primitive machine-code block by JMP (,Y++) will cause a transfer of control to the next primitive in the sequence at a cost of only two bytes and nine cycles. These powerful addressing modes also allow what I call "in-program parametrization" of any intcode command. A simple example would be DROP (*n*), a command to "drop" *n* bytes from the data stack (using the U register as the stack pointer). The parentheses signal the text-interpreter to add the numeric *n* directly to the intcode sequence, following the execution address of DROP. The code for the parametrized DROP would be:

```
DROP   LDB ,Y+ (load binary n from
         program into B register)
       LEAU B,U (reset U stack pointer
         by adding n)
       JMP (,Y++) (jump to next
         intcode command)
```

This generalized command executes in 20 cycles for any value of *n* from 1 to 127. The "reading" of the parameter also advances the Y counter to point to the *next* command in the sequence. A similar parametrization in conventional Forth would be less efficient, being sourced as *n* DROP but intcoded as 5 (instead of 3) consecutive bytes: LIT address, *n*, DROP address. Execution of LIT would (a) pick up *n* from program, and (b) save it on the data stack. Execution of DROP would (c) retrieve *n* from the data stack, and (d) use it to reset the stack pointer. Steps (b) and (c) are clearly a waste of time. There is no need for a parameter that pertains exclusively to *one* command to visit the stack at all! The stack exists to save data generated by a command for use by a subsequent command; mere copying from the program is not generation in this sense.

In-program parametrization would likewise simplify the intcode and speed up execution of the DO...LOOP construct. I shall illustrate this for Loeliger's CDO...CLOOP since this variant is more efficient for loop indices from 0 to 255. For generality, I also make the loop-increment (*i*) explicit, since this adds only one byte to the intcode and only three cycles to loop timing, and it avoids cluttering up the dictionary with distinct CLOOP and +CLOOP commands. The source code is CDO (*n m i*) ...CLOOP, where *n* and *m* are the initial and terminal indices of the loop. Here the 6809 S stack serves as the "return" stack.

```
CDO    LDA ,Y (pick up n from pro-
         gram into A register)
       LEAY 3,Y (increment Y to ad-
         dress of first in-loop command)
       PSHS Y,A (save address, then n,
         in S stack)
       JMP (,Y++) (jump to first
         in-loop command)
```

This executes in 26 cycles. Note that *n* is stacked because its value will be incremented during looping. The constants, *m* and *i*, can easily "read" by offsets using the stack-stored address.

```
CLOOP  LDX 1,S (pick up stored address
         into X index-register)
       LDD -2,X (pick up m and i into
         registers A and B)
       ADDB ,S (add i to the stack-
         stored n)
       STB ,S (save i+n, as new value
         of n)
       CMPA ,S (compare m to new n)
       BLS EXIT (if m is less or same,
         exit from loop)
       LEAY 2,X (move X plus 2 into
         Y)
       JMP (,X) (and jump back into
         loop)
EXIT   LEAS 3,S (reset S stack pointer
         before exit)
       JMP (,Y++) (and jump to post-
         loop command)
```

This more complex transfer of control costs 40 cycles for each run of the loop. Comparison with other systems would be tedious since timing information is never given, but I would be surprised if an equivalent Forth +CLOOP was as fast.

Conventional Forth allows use of different entry points into a complex machine-code routine, since all this requires is creation of a dictionary "name" that specifies the entry address. It's not clear whether multiple entry points into a complex intcode routine are also possible. In the design used by Odette, only *one* entry can exist into an intcode routine. Intcode-defined commands are, to the programmer, just "names" indistinguishable from those of primitives. If a 6809 system adopts the Odette model, the address of each secondary points to a 3-byte machine-code "leader" that is followed

by the intcode sequence. This is a JSR COLON (8 cycles) that pushes the address of the first intcode command in the sequence onto the S stack, not as a return address but for use by the COLON routine:

```
COLON LDX ,S (load new intcode
          address from stack into X
          register)
      STY ,S (save old intcode address
          in same location for return)
      LEAY 2,X  (move X plus 2 into
          Y)
      JMP (,X)  (and start running
          new intcode)
```

This takes 24 cycles, so the total entry timing loss is 32 cycles. Note that if the first two instructions were PULS X and PSHS Y the unneeded resettings of the stack pointer would add three cycles. However, resetting of the stack pointer is needed in coding the SEMI command, which terminates the secondary sequence and transfers control back to the original sequence:

```
SEMI  PULS Y (restore old intcode
          address from stack into Y)
      JMP (,Y++) (and return to
          running that intcode)
```

Since this takes 16 cycles, the total overhead of this form of high-level subroutine is 48 cycles – nearly four times longer than the machine-code JSR/RTS. It seems that the tradeoff for the conciseness of intcode is a several-fold slower execution time!

Entry to any execution address within a complex intcode sequence becomes possible if one creates a JSUB primitive, that must be followed by the "name" of the desired entry address (added to the sequence by the text-interpreter).

```
JSUB  LDX ,Y++ (load "name" address
          into X, set Y to return address)
      PSHS Y (save return address in S
          stack)
      LEAY 2,X  (move X plus 2 into
          Y)
      JMP (,X) (and start running
          "name" intcode routine)
```

Execution time is 28 cycles, only four cycles less than the single-entry-point logic, and the "call" adds two bytes to the original intcode program. The SEMI return works as before. Intcode-defined commands must be "marked" so the text interpreter will recognize that they must be preceded by JSUB. Beyond the potential advantage of using segments of a complex intcode sequence, there is forced recognition by the programmer that use (and especially nesting) of JSUB degrades timing.

One of the intriguing possibilities of in-program parametrization is in-line

# Highlights of Back Issues

## *This may be your last chance . . .*

**Response has been so overwhelming that we are out of many issues and down to only 10 or 15 copies of several others.**

**#1** **Volume I, No. 1:** Build Your Own BASIC – Design Notes for Tiny BASIC – Tiny BASIC, Extended, Part 1.

**#2** **Volume I, No. 2:** Tiny BASIC, Extended, Part 2 – A Critical Look at BASIC.

**#3** **Volume I, No. 3:** Notes to Tiny BASIC Implementers – Proposed Functions for Tiny BASIC – Denver Tiny BASIC.

**#18** **Volume II, No. 8:** Computer Applications for the Handicapped – An Interactive Programming Language for Control of Robots – A Microprocessor Operating System: The Kernel.

**#21** **Volume III, No. 1:** Memory Test for 6502 – ISIS: Anatomy of a Real-World Operating System – The SAM 76 Language.

**#28** **Volume III, No. 8:** Vocal Memory Dump – LISP for the 6800 – FORTH Dump Programs – Dumping North Star Disk Files.

**#32** **Volume IV, No. 2:** Let Your Computer Speak ASCII – Pascal Bibliography – Features for TDL Editor – An Unusual Pseudorandom Number Generator Program.

**#33** **Volume IV, No. 3:** A Critical Look at the MC68000 – CONVPATB: Palo Alto Tiny BASIC to ASCII – Falconer Floating Point Arithmetic Part 1.

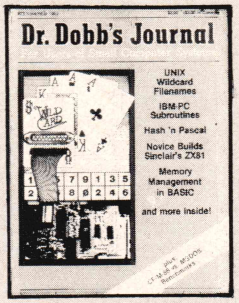
**#36** **Volume IV, No. 6:** XS-0: A Self-Explanatory School Computer – Discovering the 26th Prime – Growing, Pruning and Climbing Binary Trees with tiny-c.

**#37** **Volume IV, No. 7:** A Floating Point Subroutine Package for the 1802 – Context Switch for the 6800 – Hardware Relative Addressing for the 8080.

**#42** **Volume V, No. 2:** A User Interface to Apple-II Renumbering – TI9900: Fully Reentrant Formatter Operating Under TIPMX Part I – Converting Binary to Decimal.

**#43** **Volume V, No. 3:** TI9900 Part 2 – Mathematical Typography – A Hex Keyboard with Applications for the 1802 ELF.

**#49** **Volume V, No. 9:** Further with Tiny BASIC – CP/M to UCSD Pascal File Conversion – Extended Source-Text Area and Label Capability for SC/MP NIBASM.




**#50** **Volume V, No. 10:** N-Logs: A New Number Language – Las Vegas Super Slot – A CP/M Game – Quick-Key: A North Star BASIC Statement Generator – TUTOR: Foreign Language Vocabulary Trainer.

**#53** **Volume VI, No. 3:** Analysis of the 6502's Opcodes – ERGO: A "Thinking Program" – Computer-Based High Speed Reading – Byte Manipulation and Byte Pointers.

**#55** **Volume VI, No. 5:** Tiny BASIC for 6809's – Using and Misusing the Z-80 Microprocessor – A Smart 2716 EPROM Programmer – What to Do With Those Extra Function Keys.

**#57** **Volume VI, No. 7:** Pidgin, A Systems Programmming Language – Time on the Horizon – BASIC Rubik's, A Cube Simulator.

**#65** **Volume VII, No. 3:** Runic, An Interactive, Extensible Compiler – Pattern Classification – Pidgin for 8080 & CP/M – Fast-Branch BASIC.

**#66** **Volume VII, No. 4:** 8080-Z80/8086 Cross-Assembler, Part 2 – Writing the Runic Compiler – Poor Person's Spelling Checker.

**#68** **Volume VII, No. 6:** Multi-68000 Personal Computer – PDP-1802, Part One – Improved LET for LLL Basic – CP/M Print Utility.

**#69** **Volume VII, No. 7:** IBM-PC Issue! CP/M-86 vs. MSDOS (A Technical Comparison) – Hi-Res Graphics on the IBM-PC – PDP-1802, Part II – Review of Word Processors for IBM.

**#70** **Volume VII, No. 8:** Argum "C" Command Line Processor – SEND/RECEIVE File Transfer Utilities – Intel's 8087 – Performance Evaluation.

**#71** **Volume VII, No. 9:** FORTH Issue! Floating-Point Package – H-19 Screen Editor – Relocating, Linking Loader – Z8000 Forth – Forth Programming Style – 8086 ASCII-Binary Conversion Routines – CP/M Conditional SUBMIT.

**#72** **Volume VII, No. 10:** Portable Pidgin for Z80 – 68000 Cross Assembler, Part I – MODEM and RCP/Ms – Simplified 68000 Mnemonics – Nested Submits – 8086/88 Trig Lookup.

**#73** **Volume VII, No. 11:** Wildcard UNIX Filenames – Tests for Pidgin – 68000 Cross Assembler Listing, Part 2 – Adding More BDOS Calls – The Perfect Hash – BASIC Memory Management – Benchmarks for CP/M-86 vs. MSDOS, and the 8087.

**#74** **Volume VII, No. 12:** Small-C Compiler, v.2, Part 1 – Interrupts and CP/M – A Simple Vector-Generation Algorithm – Fifth-Generation Computers – Custom Character Sets and Lo-Res Graphics for IBM's PC – Small-C I/O Redirection and Command-Line Arguments.

**#75** **Volume VIII, No. 1:** Augusta, An ADA Subset for Micros – Xanadu: Hypertext from the Future – Stone Age Computers: 6000 Years of Computing Science – Small-C Compiler v2., Part 2.

**#76** **Volume VIII, Issue 2:** PISTOL, A Forth-like Portably Implemented STack Oriented Language – Program Linkage by Coroutines, Forth to BASIC – Linking CP/M Functions to Your High-Level Program – Concurrent CP/M-86 – Small Systems Engineering CP/M-80 Expansion Card for the Victor 9000 – REVAS Disassembler.

## TO ORDER:

Send $3.50 per issue to Dr. Dobb's Journal,
P.O. Box E, Menlo Park, CA 94025.

Please send me the issue(s) circled. 1 2 3 18 21 28 32 33 36
42 43 49 50 53 55 57 65 66 68 69 70 71 72 73 74 75 76

I enclose $_____ (U.S. check or money order). Outside the U.S., add $.50 per issue.

Please charge my: ☐ Visa ☐ MC ☐ AE Card No. _____

Exp. Date _____ Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Only the issues listed are available. Outside the U.S., add $.50 per issue ordered. Price includes issue, handling and shipment by second class or foreign surface mail. Within the U.S., please allow 6-9 weeks to process your order second class. For faster service within the U.S., we'll ship UPS if you *add* $1.00 for 1-2 issues and $.50 for *each* issue thereafter. We need a street address, not a P. O. Box. Outside the U.S., *add* $1.50 *per* issue requested for airmail shipment.

insertion of a machine-code sequence within an intcode sequence. The shift would be enabled by an MCODE primitive, defined by a simple 2-byte, 3-cycle JMP ,Y, that will transfer control to the first machine-code byte. The machine-code sequence must be terminated by a JSR SEMI, which will transfer control to the following intcode sequence, at a 24-cycle cost, for a total cost of 5 bytes and 27 cycles. This would allow use of special-purpose operations without creating new dictionary entries for them, but of course makes the program system-dependent!

The extraordinary flexibility and user-extensibility of Forth-like HLLs is a mixed blessing, since it encourages customization and the proliferation of dialects almost to the level of one-man languages, the classic Tower of Babel effect. However, I am convinced that major conceptual changes (such as parametrization) are inevitable and will sooner or later be quietly adopted. They will provide at least some common core structure and enhance the competitiveness of this linguistic type. Standardization, to whatever extent possible, should be deferred until the wealth of ideas developed in existing dialects can be pooled into an optimized core. That is how a science develops, free of orthodoxy and sectarianism, always ameliorable by common agreement, never proprietary. It should not be necessary to create scores of fully-developed rival languages such as STOIC, URTH, SNAP, or the new RPL of Stryker (1982). That is why my controversial proposals for PARFOR (*DDJ* No. 52) were purely hypothetical guidelines for possible improvement of the linguistic type, and why I have here presented only a few of the command words I have written for my 6809 system. If they're not optimal, they ought to be replaced by whatever is.

DDJ

## References


Gilbreath, J. "A High-Level Language Benchmark," *BYTE* 6(9):180, 1981.

Gordon, H. T. "PARFOR, A Theoretical Parametrized, Forth-like HLL," *Dr. Dobb's Journal* 6(2):8, 1981.

Loeliger, R. G. *Threaded Interpretive Languages*, Byte Books, Peterborough, New Hampshire, 1981.

Odette, L. "Z8000 Forth," *Dr. Dobb's Journal* 7(9):48, 1982.

Stryker, T. "BASIC, Forth, and RPL," *Micro* No.49:63, 1982.

# A Common-Sense Guide to
# Faster, Smaller BASIC

*Editor's note: The advice in this article may be highly interpreter-dependent. As most* DDJ *readers know, the inadequacies of a particular interpreter are neither universal nor necessarily long-lived. In the interest of readability and maintainability, many people deliberately sprinkle their code liberally with the very "inefficiencies" pointed out here, and later apply a mechanical compressor (several are commercially available, with varying degrees of thoroughness). Thus, two versions of each program can be kept – one for execution and one for documentation.*

In these days of increasing central processor speed and decreasing cost of memory, it is perhaps anachronistic to speak of optimizing speed and memory capabilities of your microcomputer. But assume for a minute that your home system isn't a Cray-1 with a couple of megabytes of RAM and a flock of 14" Winchesters. I remember just twenty years ago when the best military computer at sea (on the Polaris program) had a main memory cycle time of 80 milliseconds – yes, milliseconds, not microseconds. Running real-time navigation for even a submarine was touch and go. So we put eight read/write heads on one track of the drum(!), and got a 10 millisecond (Wow!) response.

**by Robert Irving**

*Robert Irving, 8637 Bothwell Road, Northridge, California 91324.*

Well, you're not that bad off. Let's assume your home system is a Z-80 and you've got 32K of RAM using a 12K BASIC. You've already got a system that is fast compared to the Polaris system, but you'd still like to get the most out of it. Here are some of the ways that you can do that. Obviously, you shouldn't expect that every one of these methods will improve speed and minimize memory usage. Some methods may do both, but most will do one or the other. In addition, some of the methods I will discuss will improve the accuracy of your results – when they can be applied.

First, let us consider speed enhancing techniques. Each individual application may appear trivial, but overall the effect is significant. Try some of the examples and you'll agree.

(1) Never put a REM statement in the middle of a set of FOR-NEXT statements.

Not this:

```
140 FOR J=1 TO 10
150 REM ** THIS IS A LOOP
160 A=A+J
170 NEXT J
```

But this:

```
140 REM ** THIS IS A LOOP
150 FOR J=1 TO 10
160 A=A+J
170 NEXT J
```

(In the first example, the REM statement would be read ten times; in the latter, only once. The more times the loop is iterated, the greater the time savings.)

(2) Avoid requiring the program to process a REM statement.

Not this:

```
270 GOSUB 7000
 . . .
7000 REM **SUBROUTINE ARCTAN
7010 REM ** ANGLE IN RADIANS
7020 . . .
```

But this:

```
270 GOSUB 7000
 . . .
6998 REM **SUBROUTINE ARCTAN
6999 REM ** ANGLE IN RADIANS
7000 . . .
```

(In the first example, two REM statements are read each time the subroutine is called. In the second example, no REM statements are ever read by the program;

they show only in the listing.)

Not this:

```
500 REM ** USER INPUT
510 INPUT "NEXT CHOICE",U$
520 IF U$="END" THEN END
. . .
550 GOTO 500
```

But this:

```
499 REM ** USER INPUT
500 INPUT "NEXT CHOICE",U$
510 IF U$="END" THEN END
. . .
540 GOTO 500
```

(In the first example, the REM statement is read each time the program is returned to the user input by Line 550. In the second example, the REM statement is ready only once in the normal course of the program.)

(3) Use special symbols in lieu of blank lines to call attention to REM statements.

Not this:

```
160 NEXT J
170 REM
180 REM FUNCTION TO BE
    PLOTTED
190 REM
200 FND(X)=4*X*X+P1
```

But this:

```
160 NEXT J
170 REM ** FUNCTION TO BE
    PLOTTED
180 FND(X)=4*X*X+P1
```

(In the first example, three REM statements must be read by the program. In the second example, only one.)

(4) Keep REM statements as short as practicable.

Not this:

```
740 REM ** THIS IS A SUBROUTINE
    TO EXTRACT THE INVERSE
    TANGENT FUNCTION
```

But this:

```
740 REM ** SUBROUTINE ARCTAN
```

(In the first example, the program must read a much more lengthy statement, even though it is ignored in processing.)

(5) Minimize display length, amount of input, and input processing for interactive programs.

Not this:

```
210 INPUT "HOW MANY OBSERVA-
    TIONS DO YOU WANT TO
    ENTER",E2
```

But this:

```
210 INPUT "NUMBER OF
    SAMPLES",E2
```

Not this:

```
1430 INPUT "NEW GAME",U$
1440 IF LEFT$(U$,1)="Y"
     THEN 230
1450 IF LEFT$(U$,1)="N"
     THEN END
1460 GOTO 1430
```

But this:

```
1430 INPUT "NEW GAME (Y/N)",U$
1440 IF U$="Y" THEN 230
1450 IF U$="N" THEN END
1460 GOTO 1430
```

(The less time spent printing unneeded displays and typing unneeded letters, the faster the operation goes.)

(6) Assign a symbolic name for frequently used constants.

Not this:

```
760 A1=3.141592654*R1*R1
770 A2=3.141592654*R2*R2
```

But this:

```
760 PI=3.141592654
770 A1=PI*R1*R1
780 A2=PI*R2*R2
```

(The interpreter reads a numerical constant character by character each time it is encountered. A constant stored as a symbolic is read character by character only once, and as a stored value subsequently.)

(7) Use simple variables (A, B, C, etc.) or nonsubscripted variables (A7, B4, C9, etc.) in lieu of subscripted variables [A(1), B(2,3), C(4), D(8,1), etc.] whenever possible. (Subscripted or array variables generally require more manipulations to store and retrieve, hence take longer.)

(8) Avoid the use of transcendental functions (trig, logs, exponentiation) whenever possible. For integral powers, use repeated multiplication.

Not this:

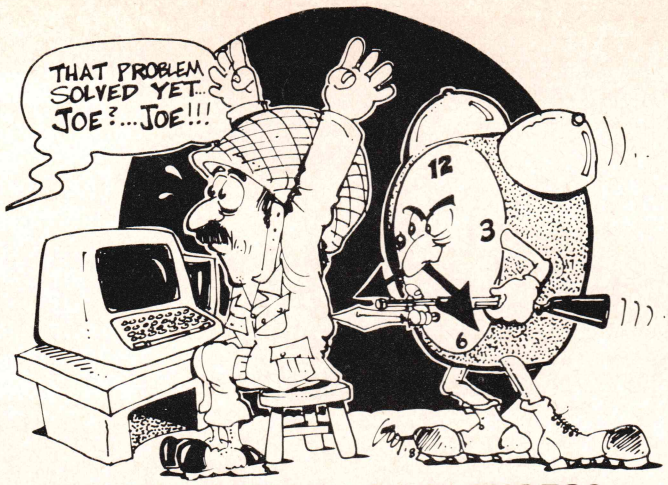
```
540 X=Y^2
```

But this:

```
540 X=Y*Y
```

(Trigonometric functions can be most time consuming. Raising a number to a power requires both LOG and EXP functions, lengthy operations, hence should be avoided for integer powers.)

Now let us turn our attention to techniques which reduce memory requirements. It is not surprising that some of the speed enhancing techniques also reduce memory. Specifically, (3) through




## *LOSING TRACK OF CHANGES?*
## "NOW! COMPARE files on CPM-80!"

- Compare Program Versions
- Fast, NO File Restrictions
- Side by Side or Vertical Listings of Differences
- Output to File or Printer
- Compare Documents
- Extensive User Options: Ignore Comments, Blank lines and more
- Create Documents with Change Bars
- Sensible Command Line defaults

COMPARE is a superb software tool with excellent features for the serious professional programmer with no time to waste. Document mode with Change Bars designed especially with Writers in mind.

Call or Write for Information ..... Only $95

**SOLUTION TECHNOLOGY, INC.**

Suite 218 • 1499 Palmetto Park Rd. • Boca Raton, FL 33432 • (305) 368-6228

*CPM-80 is a trademark of Digital Research Inc. *Check or COD, Florida residents add 5% sales tax.

Circle **no. 55** on reader service card.

## CP/M Software

**WASH**

Easy to use directory maintenance utility that replaces a dozen older programs. Menu driven for fast directory display, view or print, copy rename, delete. Also multiple copy and delete. Much easier to use than the CP/M utilities. ..... **$49.95**

**UNERA**

ERA *.BAS instead of ERA *.BAK can ruin your whole day. UNERA to the rescue — it recovers all ERAsed files for CP/M 2.2 Floppy and Hard Disk Systems with standard directories ..... **$75.00**

**FORMS-3**

Ideal for filing out all kinds of forms. Features field editing for numeric, dates, etc., justification, multipages, required entry. Can also use a separate data file. **$40.00**

**SUPERFILE**

Solves your filing problems. Menu driven information retrieval system for storing and quickly finding information. Features AND, OR and NOT in search command. Sort, merge and split utilities included. Build data base with any CP/M editor. Computer Magazine Database 900+ entries included.

**with Demo Data Base & Manual $165**
**Manual only(applies to purchase)$50**

Available 8" Single Density, North Star Single and Double Density, most 5¼" soft sectored disks.

ADD $1.50 SHIPPING AND HANDLING

CALIF. RESIDENTS ADD TAX

*Elliam Associates*

**24000 Bessemer Street**
**Woodland Hills, CA 91367**
**(213) 348-4278**

Circle **no. 56** on reader service card.

NEW

## CodeSmith™-86

**The debugger with the most bite for your IBM PC!**
**With unique multi-window/multi-level split screen features:**

- Full line-edit keyboard utilization—insert, delete, next-word, previous-word, etc.
- Full-screen disassemblies may be scrolled through with Pg up, Pg Dn, and arrow keys
- Blocks of disassembled code may be dumped to a disk file—blocks may be large, small, and/or discontinuous
- Single-step through full-screen disassemblies by hitting "+" key. Current instruction is underlined on display.
- Hundreds of simultaneous breakpoints supported. Groups of breakpoints may be tagged and toggled on/off by tag number.
- Individual breakpoints set by typing cntl-B on desired line of disassembly.
- Cntl-X starts up execution from underlined instruction.
- Saves user's graphic display when breakpoint hit, restores user's display when user's program started again. User's frozen display may be toggled to/from for observation when breakpoint hit. (Standard monochrome display subject to certain reasonable restrictions when using this feature.)
- F6 key will center disassembly display around line on which cursor is placed.
- User may type comments on disassembled code lines which will be retained throughout session
- Complete control to load your .EXE file almost anywhere in memory
- Dump display(s) may be brought up on separate windows (split-screens) or window-levels for continuous monitoring of selected memory areas. (Version 2.0 will allow user program to be running simultaneously while dump display is updating.)
- Alt-F10 key combination will interrupt program-under-test and bring up CodeSmith™ disassembly where break occurred
- Simple commands resemble Microsoft DEBUG commands, or use single-keystroke commands to speed your work
- Coded entirely in high-speed machine language
- Requires about 40K—multiple windows claim additional space on a dynamic basis
- Version 1.5 available April 1983—monochrome display version only (introductory price $145, updates $20)
- Version 2.0 (with color graphics compatibility) available July 1983 will include auto-assembly of user-entered patches, automatic label-generation option, symbolic debugging of certain PASCAL, C, and Assembler programs, 8087 disassembly and debug support, traceback of last several hundred instructions executed, watchpoint definitions (conditional halting of user program when data stored to certain locations, etc.)
- Version 3.0 available Fall 1983 will include full-screen editor option for updating your source file while still in CodeSmith™. MS-DOS Interface will be supported so that MS-DOS commands (compilations, etc.) may be entered from CodeSmith™.

***CodeSmith™ has been designed with you in mind to substantially increase your ability to get your programs working—fast***

**VISUAL AGE**

642 N. Larchmont Blvd., Los Angeles, CA 90004 (213) 464-8141

CodeSmith is a registered trademark of International Arrangements, Inc.
Microsoft and MS are registered trademarks of Microsoft Corp.
IBM is a registered trademark of International Business Machines Corp.

Circle **no. 53** on reader service card.

(7) above reduce memory requirements. There is a version of (3) which is particularly wasteful of memory; one involving the use of multiple REM statements to outline the program title, author's name, copyright statement, etc., with rows and columns of asterisks or other symbols. Each of the symbols and spaces is individually stored in memory – and that takes a lot of memory for little real benefit.

Most of the prior examples are obvious as to the reason for the saving, but a couple require explanation. In (6), added memory is needed during operation to store the value of the constant symbol – but program memory is saved since multiple symbols and not multiple values are listed in program storage. In (7), the saving in using nonsubscripted variables lies in elimination of the storage for the array definition and its addresses. These overhead functions are not needed for individual variables.

There are other memory-saving techniques:

(9) Do not overdimension arrays. Many systems have a default value (e.g., 10 or 10,10) for arrays for which you have not written a DIM statement. It may save program storage to not dimension a 3 by 4 array, but when you accept the default 10 by 10, you waste an equivalent storage for a 7 by 6 array.

(10) For systems allowing multiple statements per line, use this capability to combine short statements. The colon or other separator symbol takes less storage than a new line number. Caution: Remember that most systems skip all statements to the right of an IF-THEN in a single line, unless the IF condition is TRUE.

(11) If your system allows, use the INPUT with PRINT capability.

Not this:

```
230 PRINT "LENGTH OF SIDE"
240 INPUT L1
```

But this:

```
230 INPUT "LENTH OF SIDE",L1
```

(The latter has limits, however. You cannot insert a variable inside the INPUT with PRINT format.)

(12) If your system allows variable string lengths, set STRING= to the lowest value that will encompass the strings you plan to use. This is especially important with arrays of strings since every cell in the array will reserve as many bytes as STRING= designates. A large array using the default value for STRING= (18 bytes in some systems) can eat up a lot of memory.

(13) Avoid isolated statements – ones which will never be executed because the program bypasses them. Every programmer has, at one time or another, introduced unused material into a program. A subroutine is inserted, but never called. A GOTO bypasses one or more lines of the program. These isolated statements waste memory and should be deleted. They most frequently result from inadequate review after a program has been "patched" or rewritten.

(14) Lastly, memory can be saved by eliminating spaces in the program. Caution: Be judicious in using this technique, even if you only have a small amount of memory. A statement like:

```
100FORI=ATOM:LETJORK=TRUE:
    NEXTI
```

could possibly blow a mental fuse!

So far there has been no discussion of accuracy. Only one of the techniques mentioned so far has an influence on accuracy. In (8) avoidance of transcendentals was discussed to enhance speed. In typical BASIC systems, the TRIG, LOG and EXP functions are typically one to three places less accurate than the arithmetic functions (+, –, *, and /). Thus the use of repeated multiplications in lieu of integer powers is not only faster but more accurate.

As for fractional powers and the TAN/ARCTAN functions, in most cases one uses them in BASIC and accepts the speed and accuracy penalties. If one must have the ultimate in accuracy for a transcendental, then evaluation of a truncated series expansion is probably the best route. However, a large penalty in speed is incurred, since the series methods are usually very slow.

From the foregoing discussion, we can conclude that there are some sure ways to approach maximum speed, minimum memory use, and maximum accuracy. Not all of these methods can be applied all the time. Some methods optimize speed and memory, others optimize speed and accuracy, but none optimize all three.

DDJ

GUEST ESSAY

# A Fundamental Mistake in Compiler Design

by Edgar H. Fey, Jr.

*Edgar H. Fey, Mobile Computers, 18 W. Calendar, La Grange, Illinois 60525.*

I was stunned to read in *DDJ* that the MSDOS non-macro assembler requires 53,000 bytes of object code and the competing CP/M-86 assembler requires 27,000 bytes. I can only echo Dave Cortesi's comment, "... this kind of bloat is inexcusable."

Something is radically wrong. The authors of these assemblers are presumably competent programmers. The past records of their firms certainly indicate this. I am sure most of them are younger and smarter than I. The problem has to lie elsewhere.

In pondering this problem, I soon recognized that assemblers weren't the only culprits. In fact there is even more bloat in the typical compilers used today. There is also a fundamental reason for this bloating all down the line in typical software and the reason, I believe, is traceable to a fundamental mistake that compiler designers have been making for thirty years.

My conclusions are not based on a formal scientific proof. They are simply inferred from common sense observations. With that caveat I will first discuss bloating in assemblers and then turn to the real culprit, the conventional compiler.

## Assemblers

The state of the art for the object code size of a good structured programming machine code assembler with all the IF...THEN, IF..ELSE...THEN, BEGIN ...UNTIL constructs one needs to automatically assemble branch instructions is on the order of 1500 to 2000 bytes. The size will vary depending on the instruction set of the processor but that is roughly what one should expect.

*DDJ* published an excellent example of a 6502 assembler by W. F. Ragsdale in the September 1981 issue. Ragsdale's assembler requires only 1300 bytes of object code. I have both a 6502 and an 8080 assembler, neither quite as elegant as Ragsdale's but each is about the same size and power as his.

Do not be fooled by the small size. These assemblers are every bit as powerful as the best bloated assemblers and much more convenient to use simply because it is practical to make them resident and part of your language. You can assemble and test a machine code program on the spot, interactively, without leaving you language. I do this routinely. This would not be practical if my assembler took 53Kb or even 27Kb. My machine simply isn't that big.

The argument that since memory is so cheap, the size of a program really doesn't matter is persuasive but false. You pay a lot more than the price of additional memory for poor programming. You pay in time, convenience of use, i.e. frustration, and in the ability to understand, change and modify the program. There is a world of difference in making a simple change to a 27Kb program as compared to a 2Kb program. Further, a little bloat in an application program is of a lot less concern than bloating in the assembler, compiler and other basic tools of computing.

The state-of-the-art assemblers I have mentioned were written in the Forth language but I am sure equally good assemblers can be written in Pascal or any other good language, if someone has not already done so.

## Compilers

How big should a compiler be? I counted the space taken by the compiler functions in my Fig-Forth system. It uses only 350 bytes!

This is not one, but two orders of magnitude smaller than conventional compilers. Whether one likes the language or not is immaterial; the point is that if this is all it takes to perform the compiling functions of a powerful language (and Forth is a very powerful language), then what is wrong with conventional compilers? Where are they making their mistake?

If the compiler of your favorite language were reduced in size to 500 or 1000 bytes, you could make it a resident compiler which you could then use interactively. Wouldn't that be nice? You would then have room for the object code produced by the compiler and wouldn't need an operating system to continually swap things around.

By it being small, the compiler would be easy for you, as the user, to understand and change. You could make it extensible. Just think, if you then saw some feature you liked in some other language, you could extend your compiler to include that feature. And you could make the extension and test it on the spot, interactively, until the language was perfect for you. You wouldn't have to wait for the author of the compiler to make a set of changes that he thinks will be best for you. You make the changes that are best for you. Let him make the changes that are best for him. I call a small compiler instant computer freedom.

An extensible compiler has long been a goal of compiler writers. Most books that I have read on the subject begin with a syntax definition of languages. A particular syntax defining a language is then selected and most of the book explains how to implement the compiler based on the chosen syntax. This is really tough reading so I turn to the end of the book where I can usually find a statement to the effect: "... so that is how you write a compiler. Now it would have been nice if we could have made it extensible but that's simply not practical. All previous attempts in that direction have resulted in failure ...."

Folks, the problem of designing an extensible compiler does not lie in your techniques. They are super. The problem lies in Chapter 1:

*If you start out with a complicated syntax for a language, you will end up with a large and complicated compiler.*

It will then be too difficult for the user to extend.

One reason Charles Moore was able to write an extensible compiler for Forth is that he started out with a very simple syntax. It is so simple that it can be written in a few lines:

(1) A valid character is any member of the ASCII character set except the space and carriage return.

(2) A word is a string of ASCII characters delimited by an ASCII blank.

(3) A line is a sequence of words terminated by either the ASCII return, the established line length, or a 0.

(4) If a word in a line is

  (a) a previously defined word, it is executed.

  (b) not a previously defined word, then it must be either a number in the current base or an error.

    (1) If it is a number, it is placed on the stack.

    (2) If it is not a number, it is an error.

That's it. The language is expanded from this basic syntax. The compiler starts out very small and extensible and is kept extensible. Adding the syntax of the

usual structured programming constructs – IF...ELSE...THEN, BEGIN...UNTIL, DO loops, etc. – results in the final 350 byte Fig-Forth compiler which the user is free to expand at will.

One of the interesting products of Moore's approach is that by making the compiler small and extensible right from the very beginning, it appears that the language can be extended indefinitely. There does not appear to be an upper bound to the complexity of the language. If a user creates a syntax that appears to box him in, he should always be able to leave a path for continued expansion in some other direction.

There may be other simple syntaxes that will lead to small, powerful, efficient, interactive, extensible compilers, but the syntax used by Moore is known to work. It also may turn out that all such simple syntaxes yielding extensible compilers result in languages which have characteristics quite similar to Forth. Personally I think that would be just great, but I am sure there will be some initial objections simply because that is not the way things have been done for the last thirty years.

For example, the languages may consist of almost all verbs. Although you can put them in if you wish, there are no expository statements in Forth. Every word produces an action. The traditionalist who has been taught otherwise may say, "That's terrible – how can you express yourself in a gutteral language that doesn't have elegant expository statements?"

This is a loaded question that really has three answers:

(1) Of course you realize that you can get the computing job done. That's what verbs are all about.

(2) Do not confuse program documentation with computing. They are two separate functions. The user should not be forced to document when he feels it to be unnecessary and all he wants to do is compute. Of course it should be made convenient for the user to document to his heart's content if that is what he wishes to do.

(3) The real answer to the gutteral language charge is that with an extensible language, the user has complete freedom of expression simply because he, not the compiler writer, controls the syntax of the language. The user is in a much better position to decide what syntax he needs or would like in a particular application to express his ideas properly. He can try out a particular syntax and test it immediately. If it is unsatisfactory, he can then change it. The guiding principle is that since the bloat in compilers is related to the syntax, it is better to let the user control the syntax. The bloat will then tend to be at the top in the application and not at the bottom where it hurts.

All this talk about changing the compiler and changing the syntax must seem quite vague and mysterious to those readers who have not had the opportunity to actually do this sort of thing. It isn't. Let me try to explain with a practical example.

Say I have a file system and I wish to define individual record components in a file and give them unique names. I also want to fetch and store the record components without having to remember

their data type or their offset location in a record. Let's call the fetch and store operators V@ and V!. If the name we assign to a record component is xxxx, then we would like to be able to use the following syntax for fetching and storing data for xxxx in the current record:

xxxx V@ to fetch the component
xxxx V! to store the component

If the data type of xxxx is numerical, we will use the convention that V@ puts the fetched number on the parameter stack. If its data type is string, V@ will place the fetched string on the string stack. V! will be similarly designed. What we are trying to do is design universal fetch and store operators that receive information on the component's data type and location directly from the name of the component.

Once a protocol is set up as to how the type and offset information is to be transferred, the design of V@ and V! is straightforward. For example, xxxx could leave two numbers on the parameter stack to be picked up and used by the operators.

The compiler change comes in when we try to define xxxx. We could use the ordinary compiler to define xxxx so that when it is executed it leaves the proper numbers on the stack. But then each time we encountered a new component, we would have to tailor its definition to its data type and its offset. Sooner or later we would foul up and make a mistake. The compiler should take care of that sort of thing. So we change the compiler.

For this situation I added several new defining words to the compiler: INTEGER", BYTE", DINTEGER", and STRING", They are used to define record components in the current file, allocate space in the file records, and, when the newly defined name is invoked, pass the proper information to the fetch or store operator. It was not very difficult to do. It was all done in one session and the system was tested interactively without leaving the language.

Now when I execute

STRING" PARTNAME

I define a record component named PARTNAME in the current file. If, for example, I place the string GIZMO ONE on the string stack and then execute

PARTNAME V!

the string GIZMO ONE will be placed on the disk in the current record as the value of PARTNAME. Similarly then executing

PARTNAME V@

will fetch the string GIZMO ONE back to the string stack.

We could, of course, get fancier and have the new compilers also link the new record components to components in other files, but as you have seen, the ability to decide on a syntax change and then implement that syntax by modifying the compiler has obvious practical uses. It is a very creative form of endeavor that pays compound interest by making all subsequent computing easier.

DDJ

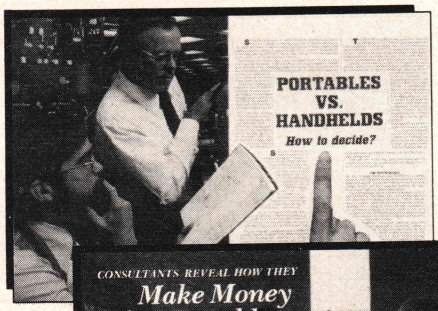
# Which portable computer is right for you?


## Introducing

# PORTABLE Computer

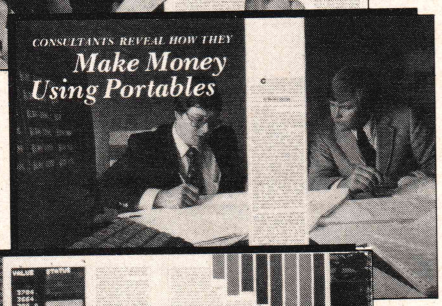
**An exciting new magazine exclusively devoted to portable computer owners (or future owners).**

***In just a few years, more than 24 million people will be using portable computers. Yet, until now, no one has provided the information people need to make good decisions about how to buy and use them. What to buy ...When to buy ...What works and doesn't work and why ...How to use the full potential of portable computers, peripherals and software... How portables can increase your business success and your personal enjoyment. Packed into every issue of PORTABLE COMPUTER, you'll find all the practical information and help you need to get more productivity, power and pleasure from your portable.***



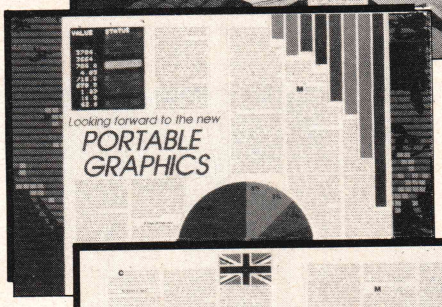

## SOFTWARE/HARDWARE

"Portables vs Handhelds, How To Decide"
"Word Processing Programs—Features To Look For"
"Where To Find The Best Spreadsheet Programs"
"The Power—And Problems—Of High Level Languages"
**You'll find this vital information plus much more, in each issue of PORTABLE COMPUTER.**




## APPLICATIONS

"Consultants Tell How To Make Money With Portables"
"Supercharge Your Salesforce With New Terminals"
"Turning On At The Hilton—Plug In To Hotel TV"
"6 Ways To Get More Accurate Estimates In The Field"
**You'll keep up with the trends...keep track of new products...keep ahead of your competition ...plus save time.**




## TECHNOLOGY & TRENDS

"Looking Forward To The New Portable Graphics"
"Adam Osborne Talks About The Future"
"Is New Bubble Memory Really Ready For The Market?"
"Folding Screens, The Solution To Small Screen Eyestrain"


## PRACTICAL IDEAS

"Tips For Traveling With Your Portable"
"Before You Buy—New Technique For Setting Priorities"
"10 Things Your Computer Store Will Never Tell You"
"Bargain Fever—Where To Get The Best Buys"
**You'll get hundreds of imaginative, practical ideas to help you make sharper decisions—and make more money—using new portables.**

***Start your subscription TODAY***

**PORTABLE Computer** 500 HOWARD STREET SAN FRANCISCO, CA 94105

## NO-RISK CHARTER OFFER FOR SUBSCRIBERS OF DR. DOBB'S JOURNAL

☐ Please bill me at the Special Charter Rate of $12.97—more than $5 off the cover price—for a year's subscription (6 bi-monthly issues).
☐ I want a special extra $1 savings, so I'm prepaying my subscription. My special price is $11.97, $6 off cover price. I still risk nothing.
☐ Enclosed is my check for $11.97, made out to PORTABLE COMPUTER.
☐ Please charge my credit card for $11.97.
☐ Visa/☐ MasterCard Expiration Date _____
Card Number _____
My signature _____
Above rates apply in USA. Outside US, add US$6.

***YES.*** Please enroll me as a Charter Subscriber to PORTABLE COMPUTER. I understand that your No-Risk Charter Offer means I can cancel after my first issue and not owe you any money.

Name _____
Address _____
_____
City _____ State _____
Postal Code _____ Country _____

Circle **no. 80** on reader service card.

# Comments on "Fifth Generation Computers"

*Editor's note: The good Doctor enjoys seeing dialogue among the readers. Recently we received the following piece in response to the guest essay "Fifth Generation Computers" by Richard Grigonis which was published in the December 1982 issue of* DDJ. *It provides some interesting counterpoints which seem likely to interest readers, and perhaps to generate more comment on the issues discussed.*

I admit that the cost-performance ratio of, say, putting an IBM 370 on a Motorola 68000 – a plan which IBM is seriously considering – should spell doom for at least the superminicomputer manufacturers. But Richard Grigonis, in his essay "Fifth Generation Computers" (*DDJ* No. 74), is not just talking about "microsuperminis" or even "micromainframes." He asks us to believe that a supercomputer, 64-bit processor running at 110 megahertz (which is essentially a Cray 2S) can be put on a single chip.

**by Michael J. Doherty**

*Michael J. Doherty, 334 South Maple Avenue, Glen Rock, New Jersey 07452.*

It will take a while for VLSI technology to catch up to mainframes in terms of performance and fault tolerance – even mainframes have fault tolerance problems. Redundant circuitry must always be built to take care of such things as processors, main memory, and recovery from faults in the I/O system. Grigonis' secondary 32-bit I/O processor, probably used for maintaining the graphics, would itself require error detecting circuitry.

The 64-bit main microprocessor would have to be so small (in order to achieve 110 megahertz) that Heisenberg's Uncertainty Principle would come into play, and the positions of the electrons in the signal pathways of the processor would become "blurred," leading to strange field effect in neighboring signal pathways. What all this means in that 90% of his processor would have to consist of error-correcting circuitry.

Granted, slower 64-bit processors will probably be in use as early as 1986. I am surprised, however, at the preciseness of his prediction that processors of *110* megahertz will have been developed by the year 1992. It would have been more realistic to suggest a barrier of 100 megahertz instead.

The gigantic 4-megabyte EEPROMS Grigonis postulates may in fact be developed by 1992, but probably for main-

frames, not micros. Even today the most advanced micro technology only provides the main memory chips used in mainframes. The microprocessor arm of this technology has been incorporated in mainframes only in the form of controllers and intelligent terminals. Microprocessor technology cannot be applied to mainframe processors because of logic circuit "randomness" problems and a general lack of performance, at least until recently. More gates than those projected for future microprocessors would be required.

Also, mainframe processor gates have switching times of about one nanosecond, as opposed to the three- to five-nanosecond switching times of the best microprocessors such as the eight-megahertz Z80-H and the 16-megahertz version of the Motorola 68000. The Grigonis 64/110 processor would probably have a switching time of about half a nanosecond – which, coincidentally, pushes silicon to its *theoretical* physical limits. I don't think a single chip made of silicon can handle the gate densities required for the operation of a processor with the Grigonis specifications.

Those 10,000- to 15,000-megabyte optical disk drives (10 inches in diameter) also give me pause. A conventional read-only videodisk can easily yield about 2,250 megabytes of storage (333,333 bits x 54,000 tracks), so Grigonis is talking about a five- to seven-fold density increase, *with* the ability to write to the disk. While I wouldn't say that's impossible, given the unknown technological developments that will occur over the next ten years, I think about 3,500 megabytes on a 14-inch disk sounds more realistic, at least as far as small systems are concerned.

Grigonis also mentions artificial intelligence programs that could require 32 megabytes of memory with a virtual memory system. Since the internal memory working set size of any virtual memory system is usually half the size of the program, he evidently envisions AI programs of 60 or 70 megabytes in length. These would be several times the size of the standard business applications packages that his AI program would presumably replace.

The problem of software quality assurance also rears its ugly head. In an actor-based, artificial-intelligence driven, customized language generator as described by Grigonis, how does one deal with program validation if the language has never before existed and the programmer is unfamiliar with it, yet is fully understood by the "machine" (meaning the actor-based AI program) that developed it? Can we be sure that the AI program really "understands" its own creation? Normally, locating errors in code that is written in a well-known language such as COBOL can take up to half of one's time – what about the unenviable position of having to locate errors in a program written in a language totally foreign to everyone?

Besides, it is my strong feeling that even with artificial intelligence, 64-bit microprocessors, and many megabytes of memory, the supermicros of 1992 will all serve merely as intelligent terminals to even more powerful mainframes, especially in the videotex environment.

Still, after years of looking at *DDJ's* countless pages of software listings in weird languages, it was stimulating to read "Fifth Generation Computers," and I would be curious to see Grigonis' speculations on the workstation environments that will have to be developed to contain the operator as well as the advanced hardware he describes.

DDJ

# YOU CAN HELP YOUR COMMUNITY LEARN ABOUT COMPUTERS

**SUPPORT**

ComputerTown™

*ComputerTown*, a computer literacy project sponsored by People's Computer Company and later funded by the National Science Foundation, began operations in 1979. The project's ongoing goal is to design, publish and disseminate materials for the development of community-based computer literacy projects around the world.

**You too can help!** Your tax-deductible donations of new and used computer hardware and software will enable us to bring "hands-on" experience to people everywhere. These items can mean the difference between a person just hearing about computer technology and actually being able to sit down and use it. A *ComputerTown* can be set up in a public library, museum, school, business or a number of other places. What's needed, along with the information, resources and experience we provide, is just a few machines and some software to get started.

Already there are more than eighty *ComputerTowns* in the U.S., Canada, England and other countries. Your generous donations can help this number grow.

For more information about *ComputerTown*, and for details on how to make contributions, contact:

Fritzi Lareau, c/o *ComputerTown*
1263 El Camino Real, P.O. Box E, Menlo Park, CA 94025
**(415) 323-3111**

# CP/M EXCHANGE

## by Gene Head

Last month this column carried the source listing for a sorted and sized directory listing program, DIR.ASM. One of the more obscure sections of that code dealt with the Disk Parameter Block and the computation of space available on a partially filled disk.

Frankly, I don't understand as much as I would like to about disk tracks, sectors, records and the like. Bob Blum has contributed two articles that he says will clear up some of the foggy parts about basic disk I/O. I'm very encouraged by Bob's contribution to this month's and next month's column for three reasons.

First, Bob is a perfect example of readers sharing their experience with the rest of us; *Dr. Dobb's* specializes in this. Second, Bob wrote the article and sent it to me via modem; our computers are helping us communicate effectively! Finally, I hope other readers will see how helpful *their* input is and send in their ideas.

Perhaps the mysteries surrounding the CBIOS to BDOS interface have discouraged you from upgrading to a higher performance disk system or from making a few of those changes that could make your system more livable. In this, the first of a two-part series, we will review the major aspects of the most popular disk drives in use today and the data allocation methods used. This will set the stage for next month's article where the disk interface portion of the CBIOS will be covered in detail. At the conclusion, I hope you will feel more comfortable with CBIOS's inner workings and consider it a friend rather than a foe.

### 3740 Format

Several years ago, IBM introduced the 3740 data entry system which used 8" floppy disks for external storage. As is characteristic of the computer industry in general when IBM introduces a new product, it is soon copied and many times becomes a new standard. This is the case with the 3740 disk format. It was the first format to be implemented under CP/M and continues to be the standard today. For that reason it will be used here as the basis for example until the discussion turns to the newer, more sophisticated disk drives.

The 3740 configuration is implemented using a single-sided, single-density, soft-sectored disk drive. Single-sided means that only one side of the media is used for recording by a single read/write (R/W) head. Referring to Figure 1 (below), each track of recording area is divided into data areas called sectors. Separation of consecutive sectors is accomplished through control information recorded along with the data portion of the sector. This type of sectoring is referred to as soft-sectored format. To reach any one of the sectors, all we need to know is the track and sector numbers. From there the disk controller and CBIOS logic direct the disk drive to step to the appropriate track and await the arrival of the desired sector. Sector skewing is employed to increase the speed at which sectors can be sequentially accessed. This technique offsets consecutive sectors from each other by a count of 6. Figure 4 (page 81) lists the standard skew table used in the 3740 standard. The skew table is used to map logical sectors to their physical location on disk. For example, your program wants to read logical sector 2. The BDOS first calls a translation routine which uses the logical sector number 2 as an index into the skew table. In position 2 is 7, offset from 1 by 6, which is used for the actual read operation. The rate at which consecutive sectors can be sequentially accessed is greatly improved by skewing because most of the time it is possible to process the data just accessed and prepare for the next operation before the next sequential sector passes under the R/W head. This makes it possible to access up to two sectors per disk revolution. Without sector skewing, the best possible data rate would be one sector per revolution of the disk.

### CP/M 3740

In CP/M 3740 single-density format there are 26 128-byte sectors per track and a total of 77 tracks per disk. Sectors are numbered from 1 to 26 and tracks from 0 to 76. The first two tracks are reserved for system use. The boot loader occupies track 0 sector 1 while the remaining sectors on track 0 and 1 are used to hold the operating system. All remaining tracks are available for directory and data area usage.

One of CP/M's major advantages is the ability to dynamically allocate disk space during program execution. To accomplish this, allocation groups are used. Each group contains eight consecutive sectors or 1024 bytes and is the basic unit of disk space which can be allocated at any one time. Control of the allocation and deallocation of groups during execution of a program requires that the current status of each group be available at all times. Memory-resident tables are used for this purpose. You have probably noticed that when logging in a new drive or warm starting, quite some time is spent in

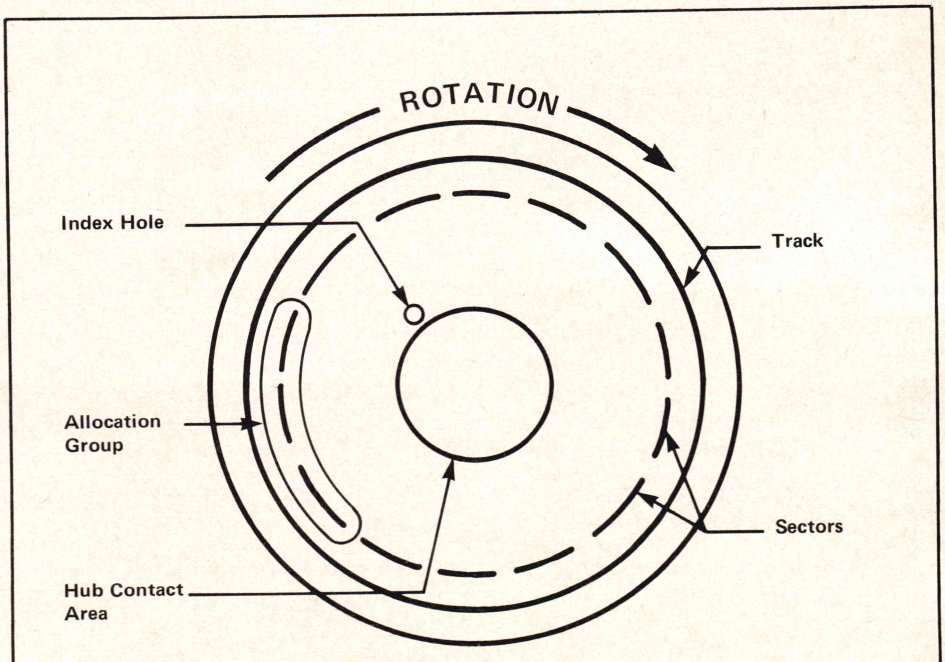



FIGURE 1.

**Master Programmer Pavel Breder Does It Again! NEW SUPER POWER! (version 3.3) puts you in control of CP/M. Now for CP/M 86 and MP/M 86, too.**

**"POWER IS A GREAT PROGRAM" - InfoWorld Software Review Nov 8/82**

# POWER!®

*The first super program that puts you in control of CP/M.®*

**POWER! works with CP/M or MP/M on any computer.**

**POWER! gives you complete control over CP/M!**

**Ever accidentally erased a file?** POWER! restores erased files!

**Ever fiddled with PIP in copying files?** POWER! replaces PIP and is faster and easier. You simply pick files to be copied from a numbered menu. POWER! feeds the names to CP/M for you - no need to type file names, no typing errors...ever!

**Tired of CP/M's scrolling through text files?** POWER! goes through files for you, page by page, file by file, or line by line with instant halt at your finger tips.

**Ever lost data on a glitched disk?** POWER! tests disks and fixes glitched disks.

**Damaged Directory?** POWER! allows you to repair the directory!

**Afraid of HEX numbers?** POWER! automatically converts HEX to DECIMAL, BINARY & ASCII.

**Need to patch or change a program?** POWER! searches memory, displays memory, and lets you change memory wherever you want.

**Want to locate a file?** POWER! sorts the directory, searches all disks or all user areas automatically for files for you.

**Annoyed at having to keep a system disk in Drive A:?** POWER! doesn't require a system disk in any drive.

**Renamed a file using = and all that typing?** POWER! lets you pick files from a numbered menu and prompts for every action.

**Ever accidentally overwritten a file?** POWER! checks first and asks permission.

## MORE THAN


**ONLY $149 ($2.75 EA. UTILITY)**

**POWER! frees your disk space since it uses less than 15k.**

**POWER! versions for CP/M or MP/M on any computer.**

**TRY IT ON US!**
**MONEY BACK GUARANTEE**

**Need to manipulate data on a disk?** POWER! reads and writes any track or sector independently.

**Ever make a mistake in the DDT?** POWER! loads disk data to ANY memory address, not just 100, and writes to the disk from any memory address. POWER! Single-Steps through memory, moves memory, compares memory sectors, tests memory, allows you to change memory and saves to disk using Decimal numbers.

***NOW POWER! permits you to securely lock any file with your password to protect sensitive information from prying eyes. PASSWORD program included FREE with every POWER! order.***

**Dislike BDOS errors?** POWER! ends BDOS errors, and gives you a way out.

**Trouble identifying files?** POWER! marks original files and their copies for you. POWER! also compares files and finds identical copies regardless of name.

**Can't remember odd file or program name abbreviations?** POWER! lets you deal with disk files by number. Never type or mistype file names again.

**POWER! does more..** NEW version of over 55 command utility programs is the only CP/M housekeeper you will ever need to really get control of your computer. A great buy, too, at less than $2.75 each.

***Previous purchasers of POWER! Exchange your original disk for updated version with the new commands and brand new manual. $35.00***
***credit card, check or C.O.D.***

**JOIN OTHER POWER USERS**

| | | |
|---|---|---|
| E. I. Dupont | Xerox Corp | AMF |
| Sperry Univac | Conn. Gen. Life | Syracuse Univ |
| NY Stock Exchange | Princeton Univ | Olivetti |
| Livermore Labs | ITT | New Mexico State |
| Union Carbide | Dow Chemical | Monsanto Chemical |
| UC Berkeley | Advanced Logic Sys. | Univ Minnesota |
| UC San Francisco | Charlston Univ | US Dynamics |
| Bendix Corp | Univ Helsinki | City Bank |
| Fort Motor Co. | Honeywell | |

**COMPUTING!** 2519 Greenwich, San Francisco, CA 94123

See Us at CPM 83 Show

**COMPUTING!** 2519 Greenwich, San Francisco, CA 94123

**TOLLFREE** (800) 227-3800 Ext 28
**IN CA:** (800) 792-0990 Ext 28
**DEALERS and OEM's** (415) 567-1634

☐ CP/M $149 ☐ CP/M-86 $149 ☐ MP/M $198 *California add 6½% sales tax.*

Card No._____ Ex Date_____
Name_____
Company_____
Address_____
City_____ State_____ Zip_____
Computer_____

Circle **no. 73** on reader service card.

disk activity. During this time CP/M is reading the entire disk directory and constructing a bit map in RAM. The bit map is appropriately named because each bit in the map represents the status, in-use or available, of one allocation group. The purpose of allocation groups is to reduce the memory requirements of the bit map and to prevent excessive file fragmentation. For example, if each allocation group were one sector in length, the bit map would contain 1944 bits or slightly over 243 bytes. Since the maximum capacity of the 3740 system is 243 allocation groups, only 31 bytes are needed to contain the bit map for each active drive. This is a large memory saving, not to mention the reduction in head movement necessary to process 1024 bytes of data. To find an allocation group in which to write, CP/M searches the bit map sequentially from the beginning until a zero bit is found which indicates availability. During the search, each bit that is passed over is counted. The resulting count is used to identify the allocation group and is used in subsequent operations to determine the track and sector numbers. When returning allocation groups to the system, the directory entries of the appropriate file are read and the allocation group numbers are extracted from the FCB and are used as an index into the bit map to appropriately adjust it.

## Double-Sided Drives

As microcomputers found more common use in commercial data processing, more capacity on the same size media created the need and subsequent introduction of the double-sided drive. In this configuration there is one stationary and one movable R/W head as in the single-sided drive. This type of drive still gives engineers fits because of unacceptable wear patterns, media warping when both heads were engaged and gravity problems since the lower head must be held in place against gravity. This also presented some new problems for the driver software and the disk controller because it was now necessary to command the drive to read or write from a particular head in addition to track and sector. One method of adapting to this new drive was to map one logical single-sided disk to each side of the two-sided disk. This achieves the desired result of twice the storage capacity but has one inherent disadvantage – excessive head movement which slows data transfer, sometimes to an unacceptable limit. The common solution was to logically view the drive as having twice as many tracks as it really had. Referring to Figure 2, it is common on a 40-track, double-sided drive to assign all tracks on size zero even numbers between 0 and 78 while side one's tracks are assigned odd numbers between 1 and 79. Since the disk controller will only recognize tracks 0 through 39, the CBIOS has to translate logical tracks to physical tracks and set the appropriate head selection line. This calculation is not too difficult. Referring to Figure 3, divide the logical track number by two. The quotient is the physical track number and the remainder is used to select the appropriate head. From this we can see that both tracks 0 and 1 are under the R/W heads at the same time and can be visualized as a cylinder. This technique provides faster data transfer because no mechanical movement is necessary to process two complete tracks of data.

Certain manufacturers weren't satisfied with only double the capacity, so double-density recording methods were developed which provided approximately twice as much data content in each of the sectors and twice the transfer rate. Today, even more data is being packed onto each disk by increasing the number of tracks and using group encoding. It's fairly common to find 640K bytes of data being stored onto one 5¼" floppy disk. With motor speed control, it is possible to store 512K bytes on a single-sided disk system as on the Victor 9000. The reward for greatest capacity on a 5¼" floppy is held by a British firm, Rair, which offers over 800K on one disk by double-sided, quad-density formatting.

Along came hard disks and still further increased storage capacity. It is not uncommon to have 200 tracks and six or eight recording surfaces provided by three or four platters on hard disks. Data transfer rates are also considerably faster because rotation speeds of approximately 3000 to 3600 RPM are common. With each movement of the R/W heads, more data are available because the cylinder has grown to multiple surfaces.

There is much more to cover in this track and sector discussion. If you have further questions, send me a SASE and I will return to you a list of reference materials which will explain these subjects in greater detail. Next month we will get straight to the heart of the interface tables and their relationship to the system.

**DDJ**

**(Figures 2, 3, and 4 at right)**

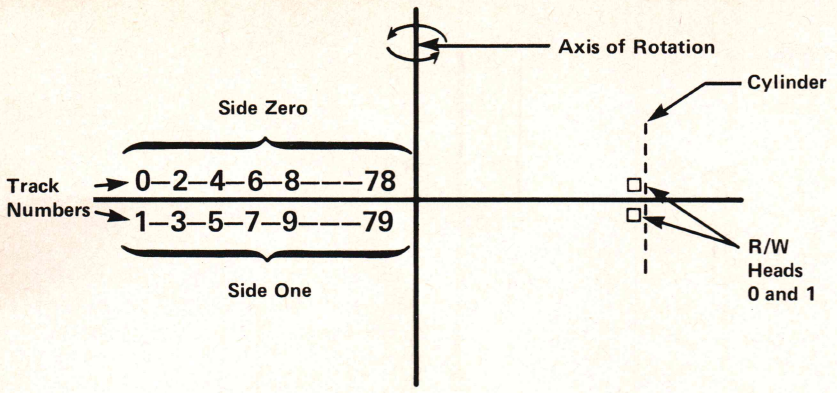



FIGURE 2.

20 (PHYSICAL TRACK)

2 | 41 (LOGICAL TRACK)
40
1 – SIDE

FIGURE 3.

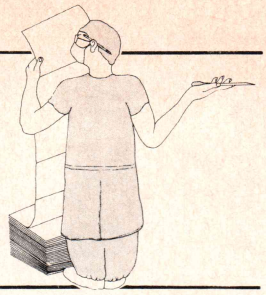
1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21, 2,
8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22

FIGURE 4.

| | Format | Code |
|---|---|---|
| 1. | 8" Single-Sided/Single-Density | 8SS/SD |
| 2. | 8" Double-Sided/Single-Density | 8DS/SD |
| 3. | 8" Single-Sided/Double-Density | 8SS/DD |
| 4. | 8" Double-Sided/Double-Density | 8DS/DD |
| 5. | 5¼" Single-Sided/Single-Density | 5SS/SD |
| 6. | 5¼" Single-Sided/Double-Density | 5SS/DD |
| 7. | 5¼" Double-Sided/Single-Density | 5DS/SD |
| 8. | 5¼" Double-Sided/Double-Density | 5DS/DD |
| 9. | 5¼" Double-Sided/Quad-Density | 5 DS/QD |
| 10. | 5¼" Single-Sided/Double-Density/Variable Speed | 5 SS/DD/V |
| 11. | 5¼" Double-Sided/Double-Density/Variable Speed | 5DS/DD/V |

Table 1.

# DR. DOBB'S CLINIC



## by D. E. Cortesi

### The Eternal Calendar

Calendar algorithms are an unfailing source of fun and agony for programmers. Early in this column's history, we proposed some calendar code. We also worked on some problems in BASIC coding whose solutions revolved around using the arithmetic value of relational expressions. Brian Moore of Oakland, CA, was going through his back issues of *DDJ* and was inspired by those old problems to propose a calendar algorithm for BASIC which uses the arithmetic value of relations.

Moore's algorithm takes a day-of-the-year number as its input, and a boolean value that is "true" if this is a leap year. In one big expression, it biases or offsets the day number to the value it would have if all months were 31 days long. The month number and the day-of-the-month number are obtained from that value by dividing by 31. Our version of Moore's code (Listing 1, below) is for the more common BASICs, in which "true" is represented as negative one and "false" as zero. Some BASICs represent "true" as positive one; if that is true of yours, change all minus signs in line 1040 to plus, and vice versa.

### The Productive Erase

Aubrey Hutchison, of Pompano Beach, FL, sends us an idea that is (we think) eccentrically brilliant. First, some background. CP/M 2.2 supports the notion of a user number, a number in 0..15 that is an implicit part of all filenames on a disk. There is a current user number (set with the USER command) which qualifies all searches of the directory, so that the only files you can see are those that were created under the current user number. It's a nice idea – it allows the directory of a large disk to be partitioned into as many as 16 sub-directories – but it wasn't carried out thoroughly enough. Later versions of CP/M (CP/M 3, Concurrent CP/M, MP/M 2, and MP/M 86) make the user number more useful.

One problem with the user number (as implemented in CP/M 2.2) is that there is no simple way to move files from one user number to another. You can do it with PIP; its [G] option will cause it to search for the source file(s) under a different user number than the current one. But since the only commands you can use are those stored under the current number, you have to have a copy of PIP under every user number you'll use. And if you want a file to appear under two user numbers, you have to have two copies of it, one stored under each number.

Well, Hutchison was pondering these matters, and thinking about the fact that the user number is stored in the first byte of directory entry as a value from 00h to 0Fh, when he had a satori of sorts. He flashed on the fact that, when the BDOS erases a file, it simply stores E5h in the first byte of the file's directory entry. So changing the user number of a file and erasing it are identical operations, namely, storing something in the first byte of the directory entry. In order to change the user code of a file, all that is needed is to persuade the BDOS to use the user number instead of E5h for erasure, and then to erase the file.

Hutchison went looking for the magic constant of E5h in the BDOS, and he found it. Just 655h bytes below the warm-start entry point of your BIOS (whose address is in location 0001h) there is an instruction, "MVI M,E5h." If that is changed to, say, "MVI M,03h" and the BDOS is called to erase a file, the file won't be erased. It'll be given user number 3 instead. Of course, the BDOS is refreshed from disk whenever a warm start is done, so the zap has to be made on the fly. You can't expect to do it with DDT, because the BDOS will be refreshed when the DDT ends.

But you could write a GO-USER command that takes a user number and a file-spec, patches the BDOS on the fly, erases the filespec, and ends. Before the command ends, it should un-patch the BDOS. That's because there are a few times when the BDOS is not reloaded from disk on a warm start – like whenever XSUB is running.

**Listing 1.**

```
1000 REM GIVEN: DAY IN 1..366, A DAY OF THE YEAR, AND
1010 REM        LEAP = TRUE (-1) OR FALSE (0),
1020 REM RETURN MONTH IN 1..12, DATE IN 1..31
1040 DI = DAY - 1
         - (DAY > (59-LEAP)) * (3+LEAP)
         - (DAY > (120-LEAP))
         - (DAY > (181-LEAP))
         - (DAY > (273-LEAP))
         - (DAY > (334-LEAP))
1050 MONTH = INT(DI/31)+1
1060 DATE  = (DI MOD 31)+1
1070 RETURN
1080 REM GIVEN YEAR IN 0..9999, RETURN LEAP TRUE OR FALSE
1090 LEAP = ((YEAR MOD 4)=0) * ((YEAR MOD 400)<>0)
1100 RETURN
```

### A Pascal Standard At Last!

Members of the IEEE Computer Society got gladsome news in the mail just before Christmas, in the form of an ad for the book *American National Standard Pascal*. According to the ad, "IEEE Standard Pascal has completed its IEEE approval and is undergoing final ANSI approval at this time . . . (it) provides an unambiguous and machine independent definition of the language." The cloth-bound book is the official standards document; it costs $17.95 to non-members, $16.95 to members of the Computer Society, and should be shipped in January. Order from IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.

We know at least one software outfit that has made frequent public promises to bring its compiler into conformance with the standard language "just as soon as there is an approved, not a draft, standard." Don't hold your breath; they'll probably claim they really meant an *international* standard, not just an American one.

### Dempsey's Dilemma

J. Dempsey, of Seattle, WA, has a sticky problem, one that most computer makers would prefer not to think about. Dempsey says, "I do writing in a technical field (linguistics) where I need to use a variety of special symbols. As far as possible, these should have equal status with the regular alphanumerics. I've seen articles on how to generate such symbols

on an IBM PC, Apple, etc. but always in a graphics, not a text, mode. [I assume that such characters] are therefore in poorer resolution than ordinary ones, clumsy to use from the keyboard, and presumably useless for word-processing or data bases. What do I do?"

What, indeed? Before you rush to answer, take a close look at one of the examples Dempsey sent along (see box, this page). It's all very well to note that the Atari, the PC, the Victor 9000, etc., all have screen character sets that can be partly or completely redesigned by the user; getting the symbols onto the screen is only a tenth of the battle.

Let's call symbols like the ones Dempsey works with *specialist* symbols: symbols not in the standard ASCII set, but which are commonplace in some specialist's field – linguistics in this case. Law, medicine, mathematics, indeed every profession, makes use of its own set of specialist symbols. The International Phonetic Alphabet is one set of specialist symbols that it would be nice to have when working with, or writing about, the Votrax speech synthesizer. Choreographers have an "alphabet" of dance movements; at one time IBM offered a Selectric typeball for it. Someone editing a dictionary would like to have a set of diacritical marks. And so on.

With many personal computers, we can put a set of specialist symbols on the screen, but only by *replacing* some part of the machine's native font. The specialist characters are folded into the "letter space" of 0..255, and so are *ambiguous* when stored in RAM or a file. A file containing them might look right when displayed on the screen (provided the right user-defined text font is loaded), but will those byte values be processed correctly by a sort? And how can they be printed? A fundamental problem is that the 8-bit symbol space is too small to accommodate more than one alphabet. This doesn't hurt when the problem is to accommodate a foreign language (although there is a problem when the foreign alphabet has a large number of symbols, as with some oriental languages). But specialist symbols are an extension of the standard alphabet, not a replacement for it.

There's a mechanical problem, too: how do you arrange a keyboard to allow for a large vocabulary of specialist characters? The usual answer is, as before, folding – some keys are preempted to stand for special codes. One of the IBM PC's best features is its ALT-shifted key values. When we first saw it, we thought it was useless, but the longer we work with it, the more uses we find for that third shift-mode.

At any rate, has any reader got either practical hints for Dempsey on what system and software to look at, or more general comments on the whole problem of specialist symbols?

## Backward with the PC . . .

The IBM PC does something very odd if you write a Backspace code (08h) from BASIC. If you write a Backspace when the cursor is at the left margin, nothing happens (which is just as it should be). But if you write a Backspace when the cursor is away from the margin, you get, not a leftward movement of the cursor, but a funny little graphics symbol, a small reverse-video diamond. Try this to see it:

```
PRINT ">"+CHR$(8)
```

Beyond the irritation of not having a Backspace that backspaces, we find this puzzling because we pored over the BIOS listing in the Technical Reference manual and we can't see where the bug is. Maybe it's in BASIC. Furthermore, the symbol displayed is not any of the ones documented in any PC manual that we can find, nor is it a reverse-video version of one of them. Can anyone explain these things?

## . . . Also Up, Down, and Around . . .

If you are using CP/M-86 on a PC, you will find that the BDOS has a rudimentary form of terminal emulation built into it, so that the PC's display can be programmed as if it were a terminal with an addressable cursor. The BDOS (not the PC's firmware) supports escape sequences to set the cursor location and to access various other features.

If you are using MSDOS, you don't have this option. Cursor addressing becomes a matter of some rather advanced PEEKS and POKES, or their equivalent in whatever language you are using. One thing that might help is to note that the ASCII control characters FS, GS, RS, and US (1Ch, 1Dh, 1Eh, and 1Fh) are defined by BASIC as moving the cursor right, left, up, and down, respectively. However, this appears to be a BASIC function, not something built into the ROM BIOS or the CRT controller chip. From assembly language, you can call the ROM BIOS directly; what you do from Pascal or COBOL, we don't know. Do you?

## . . . And BASICally Slower.

Bob Pirko, of New York, took us up on our recent challenge; he has explored the contents of IBM BASIC and demonstrated to his own satisfaction and ours that it is definitely a mechanical translation from 8080 code. He writes as follows:

"I own an IBM PC, and like you, I found IBM BASIC to be slower than I had expected. Having traced the execution of a few statements in BASIC, I can offer some reasons for lack of speed.

"Much of the code is undoubtedly a mechanical translation of 8080 code. Attached are three samples (see Listing 2, page 85) which list the code from the IBM PC's ROM, the equivalent 8080 code, and more efficient 8088 code. Table 1 (page 84) gives execution times for each of these routines.

"The IBM ROM routines are two to six times slower than they could be, and approximately twice as slow as comparable 8080 code. While such cases demonstrate the perils of mechanical translation, most of the code in IBM BASIC is not

### An example of "specialist symbols."

*jaķ nafar dehaːti bema ʃaːhr*
A peasant came to the city.

*dar baːzaːr aʃɔ*
He was going along in the bazaar.

*berasaː be dɔkkuːne ɣannaːti*
He reached the shop of a confectioner.

*ʃɯriːniːhaːje raŋɟ raŋɟ*
Confectionery of different kinds

*dɔkkun darfun tʃiː bu*
was set out in the shop.

*iːn ʃɯriːnfɔruːʃ aːhe vɛ*
This confectioner was sitting and

*aŋɟaʃt merde dehaːtiː xijaːlɛʃ ķa ɣannaːd kuːrɛ*
looking. The peasant thought that the confectioner was blind.

*dʒaldi beʃo*
He went (forward) quickly.

*merde dehaːtiː do taː uŋguːʃt dʒele*
The peasant held his two fingers before

*tʃamʃɛʃ daː*
his (the confectioner's) eyes.

*ɣannaːtɛʃ bɛʃva tʃera iːn*
The confectioner said to him, "Why did

*kaːrɛt beķe*
you do this?"

*merde dehaːti bɛʃva mɔn xijaːlem ķe tɔ kuːriː*
The peasant said, "I thought you were blind."

*mɔn kuːr nehaː avɯːnɔn*
"I am not blind; I see."

*dehaːti bɛʃva aɟar to avɯːni tʃeraː to*
The peasant said, "If you see why

*ʃɯriːnijaː naxɛri.*
do you not eat the confectionery?"

quite so bad. In fact, some of it has been optimized to use the superior features of the 8088's instruction set. Still, most routines take longer than similar 8080 code executing on a 5MHZ 8085 or even a 4MHZ Z80.

"Generally, the 8088 will execute fewer instructions per second than an 8085 or a Z80. This may seem surprising, but shouldn't be. The rate at which instructions are executed is only one of the factors which determine speed. Of equal importance is the quantity of useful work done by each instruction. Happily, the 8088 repertoire contains a large number of operations which do as much work as two or more 8080 instructions. It is only by effective use of these instructions that an 8088 can reach its true potential. Translated code is slow because it consists of just those 8088 instructions that are functionally equivalent to 8080 instructions.

"The 8080 has many one-byte instructions that execute in one memory access cycle. The equivalent 8088 code requires two bytes and often takes two memory cycles (eight clock cycles) to execute. Intel manuals list shorter times, but they assume that the instruction has been pre-fetched and is waiting in the internal queue when needed. This is often untrue, especially for the kind of code produced by a translator program.

"In summary, even if Microsoft had done a better job at optimizing the translated code, the performance would still have been below expectations. Really efficient code would have required a complete rewrite of BASIC. This would be an ideal project for someone with a couple of years available and nothing better to do."

DDJ

Table 1.

| Sample Execution Times In Micro-Seconds | | | |
|---|---|---|---|
| | IBM ROM * | Equivalent 8080 ** | Efficient 8088 * |
| 1. Add AL to BX | 8.9 | 4.0 | 2.6 |
| 2. Immediate minus BX | 10.5 | 6.0 | 5.3 |
| 3. String Move(20 bytes) | 409.6 | 188.0 | 72.9 |

* Based on time test run on an IBM PC.
** Calculated for 5 Mhz. 8085.

Listing 2.

Example 1 -- Add 8 bit value in AL to BX

| IBM ROM Code | | | Equivalent 8080 code | | Efficient 8088 code | |
|---|---|---|---|---|---|---|
| ROM Address | ROM Code | | | | | |
| F600:1893 | ADD | AL,BL | ADD | L | CBW | |
| F600:1895 | MOV | BL,AL | MOV | L,A | ADD | BX,AX |
| F600:1897 | ADC | AL,BH | ADC | H | | |
| F600:1899 | SUB | AL,BL | SUB | L | | |
| F600:189B | MOV | BH,AL | MOV | H,A | | |

This code is executed whenever one of the four arithmetic operators is specified. It is used to compute the address of the appropriate arithmetic routine. The value in AL is always less than 7FH so CBW gives the correct result. The efficient code does not preserve the contents of AH as does the ROM code. This is no problem because the subsequent code does not expect any value to be passed in AH. Indeed, it would be surprising if it did since translated code doesn't know that the AH register even exists.

Example 2 -- Subtract BX from an immediate value (0FF26H)

| IBM ROM Code | | | Equivalent 8080 code | | Efficient 8088 code | |
|---|---|---|---|---|---|---|
| ROM Address | ROM Code | | | | | |
| F600:2CE0 | MOV | AL,26H | MVI | A,26H | NEG | BX |
| F600:2CE2 | SUB | AL,BL | SUB | L | SUB | BX,0DAH |
| F600:2CE4 | MOV | BL,AL | MOV | L,A | | |
| F600:2CE6 | MOV | AL,0FFH | MVI | A,0FFH | | |
| F600:2CE8 | SBB | AL,BH | SBB | H | | |
| F600:2CEA | MOV | BH,AL | MOV | H,A | | |

This is executed once for each variable in a BASIC statement. I think this routine checks whether the stack has overrun its limits. I used SUB BX,0DAH rather than the more natural ADD BX,0FF26H in order to set the flags the same as the original code.

Example 3 -- String move

| IBM ROM Code | | | Equivalent 8080 code | | | Efficient 8088 code | |
|---|---|---|---|---|---|---|---|
| ROM Address | ROM Code | | | | | | |
| F600:2895 | INC | BL | | INR | L | | |
| F600:2897 | DEC | BL | MOVEIT: | DCR | L | MOV | SI,CX |
| F600:2899 | JNZ | 289CH | | | | MOV | DI,DX |
| F600:289B | RET | | | RZ | | MOV | CL,BL |
| F600:289C | MOV | SI,CX | | | | XOR | CH,CH |
| F600:289E | LODSB | | | LDAX | B | REP | MOVSB |
| F600:289F | MOV | DI,DX | | | | MOV | DX,DI |
| F600:28A1 | STOSB | | | STAX | D | RET | |
| F600:28A2 | INC | CX | | INX | B | | |
| F600:28A3 | INC | DX | | INX | D | | |
| F600:28A4 | JMP | 2897H | | JMP | MOVEIT | | |

This routine is used by BASIC to move string variables. It shows how bad mechanical translation can be. The 8080 uses BC and DE as memory pointers. The Intel standard is to equate BC and DE with CX and DX respectively, and the ROM obeys this convention. Since CX and DX can't be used as memory pointers the ROM must add instructions to move the addresses to SI and DI.

# OF INTEREST

## by Michael Wiesenberg

### Move Over Ada, Here Comes Beb

The Department of Offense has announced Beb, their new programming language named after Mary Shelley's maid's daughter, B'eb, now conceded to have been the world's first computer programmer. B'eb is reliably documented as having programmed an idiot savant to balance her checkbook in 1829, three years before Lord Byron's daughter, Ada, even met Charles Babbage (and, in fact, several years before checkbooks were even in the hands of the general public). Beb uses the IF . . . THEN . . .ELSE . . .BUT ON THE OTHER HAND construct. A RANDOM GOTO statement randomly jumps to any program line; its EXTERNAL option causes a jump to any random memory location, including some not accessible to the processor. Beb saves program space by writing all data to code space. Not only does Beb have no I/O routines, it does not permit them to be written. Beb incorrectly executes subroutines written in any other programming language. Optimizing routines reduce any source program to exactly three bytes of object code. Conditional compilation causes the computer to compile your program only if it feels like it. Nested infinite loops lock up the CPU, bringing any time-sharing system to its knees. Touted as the first truly portable language, Beb can be installed on any electrical appliance, including such popular machines as VersaTeller and all Toshiba microwave ovens. **Reader Service No. 3.14159265.**

### A Trendy Giggle

Here's a new word for you to practice: *gigabyte*. It means "one billion bytes." That is, a thousand megabytes. It's pronounced with a hard *g* followed by a short *i*. I've been reporting disk drives with ever more storage, up to 160 megabytes and counting. We recently received notice from **Capital Systems** that they will be a distributor of **IBIS Systems** IBM-plug compatible 1.25- and 2.52-gigabyte drives for mainframes, primarily for sale to U.S. government agencies. Not for the personal computing crowd yet, nor were prices even quoted, but soon you'll be seeing a lot of that word. And remember the abbreviation *GB*. (Or should it be "Gb," to be consistent with "Mb" for megabyte?) And try to think what you could *do* with all that storage. . . . IBIS also makes a disk drive controller for drives with storage capacities up to *40 Gb*! (And don't ask me if that's *really* a billion, or 1024-cubed. . . .) **Reader Service No. 101.**

### A Wonderful Software Marketing Trend

**BLOWUP**, written in BASIC for the IBM PC, transforms text into inch-high (ten characters by ten lines) block characters on a printer for signs and notices. Not only does the 5¼-inch diskette retail for only $24.95, but it introduces an unusual but welcome distribution philosophy to software marketing that I hope becomes a widespread trend. **Robert A. Murray and Associates** encourage you to copy the software and give it to friends, with the philosophy that "good, inexpensive software will sell itself." They ask only that those who find the software acceptable send them $20, in return for which they will receive a free catalogue of other Murray software products. The honor system: what a refreshing answer to piracy! **Reader Service No. 103.**

### Color It Any Color

**USI** has introduced a 14-inch color monitor with a high resolution, 80-column, 24-line display in seven colors that retails for $399. This is a vast improvement over a TV set, which is not made to be a computer monitor, its low resolution producing blurry characters and fuzzy graphics. USI last year gave this country its first amber screen monitor. Both monitors, and a green phosphor model, are available in 9- or 12-inch sizes. Each USI monitor is housed in a stackable metal case and is style- and color-coordinated with most popular computers, to which they connect with standard phono jacks. **Reader Service No. 105.**

And for a screen dump to printer of your wonderful color displays, the **Transtar 315** prints in color for the same price as many black-and-white dot matrix printers. For $599 you get a four-hammer print head and the ability to print seven colors plus more than 30 shades in a single pass. Color graphics and characters both print at 50 cps. Print speed has little meaning in many printers that intermix colors in multiple passes or multiple strikes at the same spot, but the Transtar 315 does not slow down, pause, or come back. And thus throughput "exceeds machines rated at 200 cps." You can get an interface option right now for Apple][ (and for other computers soon) called **PICS** that lets you press a copy button on the printer to get a direct color graphics dump of the screen without using disks, exiting programs, or changing application programs. Transtar also offers a daisy wheel printer, the **Transtar 130**, for $895, that immediately runs from most micros with all word processing packages that have Diablo print routines. You get boldface, underscore, proportional and incremental spacing, parallel or serial interface (one or the other), 300 to 2400 baud, 16 cps throughput, auto paper loading, bi-directional printing, and a six-month warranty. There is also a **Transtar 140**, for which they didn't quote a price, that has all of the preceding, plus 40 cps and a bidirectional tractor feed option. (The printers, by the way, are all made by Seiko's Seikosha Group.) **Reader Service No. 107.**

### Adding to Your Color Computer

Enhance BASIC on your TRS-80 Color Computer with **Spectrum Projects' Basic Aid**, a ROM cartridge that sells for $39.95. Automatic line numbering, single-key entry of most commands, redefinition of keys, merging routines stored on tape into a program in memory with automatic renumbering so tape libraries can be built, moving of any program lines anywhere about a program with consequent renumbering of all GOTOs, GOSUBs, and so on, that reference the moved section, are all part of Basic Aid. **Colorcom/E** turns the Color Computer into a smart terminal with on- and off-line scrolling, off-line printing, transfer of cassette files, serial printer support and full or half duplex. The program costs $49.95. They also sell a **Spectrum Stick** joystick that has a swivel ball, long cable, sturdy construction and red LED on-indicator for $39.95 plus $2 S&H. They offer cables that extend the ROMpack port by three feet, add 10

feet to the joystick cable, extend the cassette recorder or printer/modem cable by 10 feet, and connect in line both a joystick and their newest product (so new, they didn't even give us a price, but write them), a light pen. An RS232 expansion cable attaches two devices to the serial port at the same time, permitting a printer and modem to be attached in line, for $19.95. You can use your new equipment to access Spectrum's two bulletin boards, and Colorcom/E to download color graphics: (212) 441-3755 and (212) 441-3766. **Reader Service No. 109.**

## The IDE of March

**IDE Associates** makes a 3.9-inch Winchester disk with fixed or removable cartridges for the IBM PC. Formatted capacity of 5.0 Mb costs $1450 for the internal mount-fixed cartridge version. The removable cartridge versions cost $50 more. The price includes mounting hardware, cables, software and diagnostics diskette, and manual. IDE offers, in ten metropolitan areas, what they claim is "the industry's only free on-site installation service." IDE also sells expansion memory and clock cards for the PC. For all their products, you get a one-year return-to-factory warranty, or, in any of the ten areas, the option to convert the warranty to a fixed-price on-site maintenance agreement. **Reader Service No. 111.**

## More Memory at Less Price for Chipmunks

Expand memory on HP's Series 200 (the 9826 and 9836 68000-based computers known affectionately in house as "Chipmunk," and the new 9816 personal "home" computer) with **Eventide's WKBP-4** 256K memory expansion board. At $749, it costs "hundreds" less than HP's equivalent. You can add half a megabyte to the 9816 for under $1500, and over two megabytes to the 9826 and 9836 for, they say, "less than $5200." I'm not sure how they perform this mathematical feat, unless they offer discounts for eight boards . . . . **Reader Service No. 113.**

## Altos Talks to Other CP/Ms

**InterComm** from **Acquis Data** transfers files between an Altos and other CP/M computers. Included is a module to communicate with bulletin boards and public data bases like CompuServe, Dow Jones, and The Source. For $175, you get two diskettes, manual, and communications/null modem cable. InterComm is also available for many other CP/M computers. **Reader Service No. 115.**

## If The Code Works, Use It

Version 2.0 of the **Q/C** compiler for CP/M, from **The Code Works**, supports a large subset (all but float and long data types, multidimensional arrays, structures, sizeof, typedef, and casts) of the C language, compatible with Bell Labs' UNIX 7 C, and includes a library of over 50 I/O functions. Q/C produces true native code for Z80 or 8080 for Microsoft's M80 assembler or for Digital Research's ASM or MAC. Assembly language functions can be included in programs. With M80, programs can be compiled as separate modules and Microsoft's L80 linker can be used to combine the relocatable files. Full source code for compiler and library are included, as well as a comprehensive, beautifully typeset 138-page manual. You'll need a 56K CP/M, and either soft-sectored 8-inch single-density or hard-sectored 5¼-inch Micropolis Mod-II format double-density (Vector Graphics MZ) disk. The Q/C disk costs $95. Look for version 3.0 soon, which

# JANUS ADA*

# Take A Test Drive!

We all know how important the test drive is when choosing a car. But how do you choose the right language for your programming needs?

Now we've just made it easier for you to make the right choice. Our new demo package allows you to experience the power of **JANUS/Ada.**

**JANUS/Ada** is a subset implementation of Ada that includes many features not found in any other micro-processor programming language. These include true modular programming, full error messages in English, error walk-backs, and re-entrant initialized variables. These and more features are described in greater detail in our informative brochure.

Take up to 30 days to experience the power of **JANUS/Ada**. Make sure it does what you want. Then if you find it isn't right for you, send it back and we'll return your money, no questions asked. But we're sure you'll want the complete package after experiencing part of the power of **JANUS/Ada**. Best of all you can get the demo package at the introductory price of $30.00. This offer concludes after the West Coast Computer Faire, March 31, 1983. Drop by the Faire and see us at our booth.

*The language that is based on the past but looks to the uses of the future.*

**Information**
Call, write or circle our reader service number to receive our informative brochure.

**Ordering**
Please specify your microcomputer, CPU, disk format and operating system.

**JANUS/Ada Demo Disk and Manual**
Contains evaluation compiler, linker and example programs.
Available on 8" MS-DOS, 8" CP/M, Apple softcard and IBM-PC . . . . . $30.00
$30.00 can be applied to full **JANUS/Ada** package.

**JANUS/Ada Package**
Contains complete compiler, linker, assembler, example programs, manual and more . . . . . Prices from $300
Available on most disk formats. Call for your system price.

CP/M, CP/M-86, MP/M-86 are trademarks of Digital Research, Inc.
* ADA is a trademark of the U.S. Department of Defense
MS-DOS is a trademark of Microsoft
Apple Softcard is a trademark of Microsoft, Inc.


RR SOFTWARE

*specialists in state of the art programming*

P.O. BOX 1512 MADISON, WISCONSIN 53701 (608) 244-6436

Circle **no. 79** on reader service card.

will support structures, unions, multi-dimensional arrays, and a few other new features, using the Z80 instruction set, and producing smaller, quicker code. This will be offered as a $12 upgrade to current users. A version for the IBM PC (PC DOS) will also be released at that time. The Code Works also offers what they call the "original" **Small-C** by Ron Cain for $19.95, a much smaller subset of C, with a minimal assembly-language I/O library and an 11-page manual. It needs 48K. Add $3.50 shipping in U.S. and Canada, or $15 elsewhere, and, in California, 6% sales tax. The Code Works charges no license fees for programs created on its compilers and libraries. **Reader Service No. 117.**

## Flying a Simulator? Simulating Flying?

Now you can do your armchair flying – from the armchair facing your computer, or flying by the seat of your pants – without leaving your terminal. **Flight Simulator** by **Microsoft** for the IBM PC is for novice to experienced pilots. As your skill increases, you add various conditions, such as time of day, cloud cover, season, wind shear, trim, carburetor icing, fuel usage, and even radio navigation. The screen displays an "out-the-window" 3-D perspective view with full instrumentation. You have real-time control of the plane (a single-engine Cessna 182) in several flight conditions, and data that simulate twenty-three airports in four parts of the country (Seattle, LA, Chicago, and New England). You also get **British Ace**, a World War I combat game that test your flying skill and strategy against enemy biplanes defending territory that you must bomb. You'll need a PC with PC-DOS, color/graphics card, 64K, one disk drive, and a monitor (they recommend the standard composite color monitor, but you can use black-and-white or RGB). $49.95. **Reader Service No. 121.**

## Legal Copy Service

**Port-A-Soft** (formerly Disk Copy Service) converts disks between several formats, to or from 5¼- or 8-inch, single-, double-, or quad-density, in formats of various operating systems and many popular computers, for $5 to $15 per diskette, and copies after conversion for $2.50 each. **Reader Service No. 119.**

## Other Stuff

Do you own or have access to a DEC system with RT-11, TSX, or TSX-Plus? Do Z80 software development on it with **SATEC Systems' Z80 Cross Assembler**, available for $200 on paper tape, 8-inch floppy, or source listing. **Reader Service No. 123.**

Here's your chance to see if you are smarter than a computer at word games. **Wordtrix** from **Insoft** for the IBM PC has you compete against the machine to find words in a random grid of letters. The computer brings its own dictionary (how can you argue with the guy who brings the bat?) and plays on several levels. $34.95. **Reader Service No. 125.**

Learn about protecting software, copyrighting programs, writing and negotiating your own licensing contracts, limiting liability, and other legal considerations for programmers and software publishers in *Legal Care for Your Software* by Attorney Daniel Remer, published and distributed by **Nolo Press. Reader Service No. 127.**

DDJ

dtoi unnecessarily requires two (as do Hendrix's utoi and xtoi). Also discordant with K & R's atoi function, Mr. Hendrix's itou and itox also require one more parameter than they should. The string size parameter is useful but, probably more often than not, not necessary. Perhaps these functions should be retained, given names to denote their special ability, and not published as the main functions of their type in a "standard" library.

I consider the loose interpretation of the C standard as it pertains to its libraries a major problem for its future portability. I've tried several of the compilers available commercially and found them all to have problems in this area.

Both the BDS-C and the Supersoft C compilers have non-standard fopen functions. That's one function they should know enough to leave alone. BDS doesn't implement the mode feature at all, and requires a pointer to a buffer be passed to it (why?). Supersoft simply adds an additional argument allowing you to specify your buffer size. Handy perhaps, but non-standard! Why not make it an additional function by a different name if you need the "enhancement." Ah, and what about the Cadillac – Whitesmith's? It too requires an additional argument to specify that the buffer address be passed to its fopen. But that's just the beginning for Whitesmith's.

Their enhancements (read "incompatibilities") are legion. Believe it or not, they don't even have a printf function! Or a scanf. They have putfmt and getfmt instead. They are slightly more flexible, and don't use the old standard names which is good, but why no printf and scanf? Whitesmith's list of small "enhancements" goes on and on: strcpy is called cpystr, strcmp is called cmpstr, atoi is called btoi (buffer to int), itoa is called itob, etc. My pessimistic side says they do this to lock you in to their family of C compilers (8080, PDP-11, 68000, etc.).

Come on guys, it's a great language; don't muck it up.

Anyone even halfway serious about using C and perhaps building his own extended library (that's what it's all about) should read the K & R book from cover to cover at least twice, and pay close attention to the style the fathers of C have developed and to their many good examples. Their's is only one way of doing it. But it's a good way, and it's the standard.

Not to detract from the very important C problem, I'd like to make a note in reference to the "Interrupts and CP/M" article. Circular or ring buffers, as they're called, are very natural with most processors, and very handy as the article shows. I think Mr. Bromberger may have missed the point that many things in the computer world are naturally circular. As an example, his code to keep the buffer pointer within the limits of the buffer:

```
        inr
        cpi
        jc
        mvi
int1:
```

can easily be replaced with the following code which is smaller and, I think, more indicative of the buffer's circularity.

```
        inr     a
        ani     7fh
int1:
```

This requires that the buffer size be an even power of two, and the mask (7fh in the example) be one less than the buffer size.

I'd also like to mention that I feel that little flap with the JRT people is a good reaffirmation of *DDJ's* separation of editorial and advertising policies. Too bad for JRT, they really did us all a service by starting the $29.95 movement.

Thanks for a great magazine, and to these authors in particular for some very accessible and useful information.

Sincerely,
Richard Foulk
Pegasus Software
P.O. Box 10J
Honolulu, HI 96816

## What's the Holdup?

Dear Editor:

JRT Pascal seems to be a current topic of interest among owners of micro computers. There have been several letters published reporting problems with the $29.95 Pascal system. Maybe this will be a new activity to be engaged in. I am sure that the problems that the owners have found with JRT Pascal are real. If JRT Systems is not going to solve these problems, I feel certain that the readers of *Dr. Dobb's Journal* will soon be reporting their solutions. Since the JRT System is priced within the reach of every computer owner, there will be many working on meeting the new challenge.

I have a problem that is not new with those of us who order by mail. I had my order processed on September 22, 1982 by JRT Systems. I have not received my JRT Pascal as yet. Several letters to JRT Systems have gone unanswered. I hope that my program arrives soon, and then I can start finding if JRT Pascal is good or BAD.

Sincerely,
Donald M. Dealy
231 Washington St.
S. Attleboro, MA 02703

*Ed Note: The preceding letter was not the first that we have received of its kind. Since a number of you may be wondering where your package is, we thought that you might be interested in the following letter which we recently received from JRT Systems concerning late deliveries. We see that the letter does not mention adjustments to any of the bugs that have been noted, but we understand that JRT has a new version which they expect to begin shipping in mid-February.*

Dear Editor,

We are having delays in the shipping of JRT Pascal to our customers. I have received a number of letters from customers who are very concerned about this. Similar letters have been sent to the editors of the magazines in which JRT Systems advertises.

It has always been our policy to immediately cancel any order on request or make an immediate refund if payment has been processed and shipment not yet made.

As of today we have shipped 10,000 Pascals and have 7,000 orders on backlog. About 6,000 of those are less than six weeks old. A six to eight week shipping delay is not unprecedented in the computer industry, but it is not acceptable to JRT Systems.

How did this backlog develop and what am I doing to correct it? In May of 1982, I cut the price of JRT Pascal from $295 to $29.95. I tested this formula by "mass" mailings of 100 then 400 then 1000 brochures. Sales response was excellent – 10% to 15% – on the early mailings. I was able to very rapidly expand the mass mailings and advertising leading to these approximate sales figures: May - 100, June - 350, July - 700, August - 1400, September - 2000, October - 4000, November - 3000, December - 5000.

In August JRT Systems moved from my home to a small office on Irving Street in San Francisco. By October this office was severely crowded with six people, three phones and two computers. In mid-December we leased 6,000 square feet of space in Mill Valley. The shipping department has now moved from Irving into 1,600 square feet of the new space. In the past two weeks, the shipping staff has grown from one to four full-time people. In this same period, we completed installation of a sophisticated set of new shipping programs which automates, logs and validates every aspect of the shipping operation. Last week we exceeded 500 Pascal shipments per day. With our new system we can exceed 1000 per day.

We still have delays in obtaining copied diskettes rapidly enough, especially in 5¼" formats. Changes planned for the near future will eliminate this problem.

In short, the delay in shipping JRT Pascal is due to the staggering sales growth. We are moving as fast as possible to expand production capacity.

Sincerely,
J. R. Tyson, President
JRT Systems
550 Irving Street
San Francisco, CA 94122
(415) 566-5100

## A Few Suggestions

Dear Editor,

First, I want to address the contents of the columns, Of Interest, CP/M Exchange, and 16-Bit Software Toolbox. The first of these columns contains information which I think most *DDJ* readers find redundant. I think that *DDJ* should devote its space to articles and not to condensed press releases which may be found in *BYTE* or any number of other journals. Conversely, I think that the second and third columns have a place in *DDJ*. However, I think that they need some changes. CP/M Exchange needs to deal with topics, not just contain announcements. I would like to see the columnist find topics of interest and discuss them instead of continually referring to RCP/M related announcements. Finally, I think that 16-Bit Software Toolbox should stay away from press releases and concentrate on one (or at most two) topics per month. This would allow the columnist to explore the topics more carefully.

Second, I would like to reiterate my comment concerning the overall quality of the journal. I strongly believe that good material still appears in *DDJ*. However, it is wise to remember that good material appeared in quantity even when the magazine ran with no advertisements. One would expect quality to increase and not for mediocrity to become a serious threat once ads were included.

Finally, I would like to make one additonal comment. I think that articles should be rated by user reaction, and remuneration based thereon. This is already done by *BYTE*.

Sincerely,
Anthony Skjellum
1695 Shennandoah Road
San Marino, CA 91108

## Keeping Us Honest

*Joe Williams called in to let us know that we had a minor error in the Small-C compiler listing in the January 1983 issue. The listing portion at the top of page 63 should have preceeded the listing portion on the top of page 62, not followed it. – Ed.*

DDJ

# COMPUVIEW'S CP/M-86 GIVES YOU WHAT IBM CAN'T

## Increased Productivity

Improve your productivity with built-in horizontal scrolling (254 columns) and screen line editing. This lets you extensively edit or re-enter any command line on the screen and greatly reduces the amount of re-typing necessary due to mis-typed or repeated commands. It's almost like having a built-in full screen editor for every program you use. And with 25% more disk capacity you will be swapping disks a lot less.

## We Don't Lock You In

Read and write not only IBM CP/M-86 disks, but also IBM MSDOS and other CP/M double density disks. Transfer files with other CP/M and CP/M-86 computers via the serial port. The screen driver with status line emulates many popular terminals. And of course we're software compatible with IBM.

## Winchester Disk Support

Special versions available to support the TECMAR, DeVong, Corvus and other hard disks. Or have a quad density 96 tpi double sided floppy as drive B (or C and D externally) with 782K capacity.

## No Software Shortage

Most CBASIC programs run perfectly with our CP/M-86 and CBASIC-86. Even most programs compiled with CBASIC 8080 will run with CBASIC-86. And Pascal-MT is available too.

### Compare CompuView with IBM CP/M-86

| Feature | Compuview | IBM |
|---|---|---|
| Horizontal Scrolling | Yes | No |
| Screen Line Editing | Yes | No |
| Page Control | Yes | No |
| Emulate Popular Terminals | Yes | No |
| 'Smart' CRT Functions | Yes | No |
| Read/Write IBM MSDOS Disks | Yes | No |
| Serial File Transfer | Yes | No |
| Support Non-IBM Hardware | Yes | No |
| Menu Driven Configuration | Yes | No |
| Programmable Function Keys | Yes | Yes |
| Status Line | Yes | Yes |
| Serial and Parallel Printers | Yes | Yes |
| File Capacity | 193K | 154K |

CP/M-86 for IBM PC .....$285
Quad Density Drive Version .$350
Winchester Disk Version ....$425
Manual Only .....$20

VEDIT-86 with above purchase. This version of VEDIT has horizontal scrolling (254 columns) ..$125

CBASIC-86 .....$325
PASCAL-MT-86 .....$600

Tandon double sided 80 track drive gives 782K file capacity. Fits into IBM PC as drive B, or connect two externally. CompuView CP/M-86 required. .....$450

### More Storage Lower Cost Than IBM

IBM Double Sided
40 Track Drive .....$650
IBM CP/M-86 .....$280
Total (320K file capacity) ...$930

Tandon Double Sided
80 Track Drive .....$450
CompuView CP/M-86 .....$350
Total (782K file capacity) ...$800

## V-COM DISASSEMBLER
## Labels, ASCII, Exceptional Speed

No other Z80 CP/M disassembler produces understandable source code as quickly as V-COM. It is INTEL and ZILOG compatible, and features easy to read code with a cross reference table. Best of all, it can create source code with user defined labels, storage areas and ASCII strings. V-COM is exceptionally fast and can disassemble a typical 12K .COM file into a 76K .ASM file, containing 7500 lines of source code, and a 33K cross reference file in under two minutes with 8" SD floppies. (About five times faster than others).

You can create two auxiliary files to easily specify labels for 8 and 16 bit values and the location of storage areas, tables and ASCII strings. The disassembled code can be sent to the console, the disk and the printer, or any combination at once.

Each package includes a 30 page manual, sample program files and variations of V-COM compatible with TDL, MAC and ZILOG assemblers. Feature for feature, no other disassembler at any price even comes close. .....$80
Manual only .....$12

## 8086 SOFTWARE

**NEW** Terminal and File Transfer program for IBM PC, Displaywriter and other CP/M-86 and MSDOS systems. (Superset of MODEM7) .....$70

VEDIT full screen editor for CP/M-86, MSDOS, IBM Personal Computer and IBM Displaywriter. .....$195

CP/M-86 BIOS for popular S-100 disk controllers and SCP 8086 computer. Source Code. .....$90

Bootable CP/M-86 disks for popular S-100 computers ..Call

# New Software from CompuView

## Mainframe Features for Microcomputers

## MODEM-86 Communications for CP/M-86 and MSDOS

MODEM-86 is the first truly universal communication program. It allows you to access a dial-up computer, capture and store the data on disk, or transfer files back and forth (using X-ON/X-OFF). Single and multiple files (both ASCII and Binary) may also be transferred reliably with error checking/correction between any system running MODEM86 or the popular MODEM4 and MODEM7 programs. The help command, command menu (expert mode turns menu off), and directory display simplify operation.

The unique installation supports the IBM PC and Displaywriter, other popular 8086 computers and many S-100 I/O boards. Finally you can communicate with almost any other computer.

Version for CP/M-86 or MSDOS .$89
For both CP/M-86 and MSDOS .$120

## COMPUVIEW ADVANCED CP/M-86 FOR IBM PERSONAL COMPUTER

Advanced features include built-in horizontal scrolling and screen line editing. Includes ability to read/write IBM CP/M-86 and PCDOS disks, emulation of popular CRT terminals, a menu driven configuration, higher disk capacity and serial file transfer with other computers. Special versions are available to support 80 track drives, TECMAR, DaVong and other hard disks.

CP/M-86 for IBM PC ..... $325
Winchester disk version ..... $425

V-DISK - An extension to our CP/M-86 intended for software distributors. Allows production of common double density disks (Televideo 802, DEC VT180, NEC PC8000, SuperBrain, etc.) on the IBM PC ..... $500, plus $40/format

## V-SPOOL - 16K Software Print Buffer

Instantly buffers up to 16K of text destined for the printer in memory. Instead of waiting for the text to print, you retain complete computer control while the buffered text is sent to the printer. Never loses your keystrokes! Your time savings will be substantial, and the operation as simple as a single command. Requires no hardware or software modifications, just CP/M 2.2. Occupies only 3K of memory plus the size of the variable print buffer ..... $79

## BIOS FOR CP/M-86 AND MSDOS

Call for details on CP/M-86 BIOS for popular S-100 disk controllers (track buffering available) and MSDOS BIOS for hard disks and CompuPro disk controllers.

## V-COM DISASSEMBLER Labels, ASCII, Exceptional Speed

No other Z80 CP/M disassembler produces understandable source code as quickly as V-COM. It is INTEL and ZILOG compatible, and features easy to read code with a cross reference table. Best of all, it can create source code with user defined labels, storage areas, and ASCII strings.

Exceptionally speed - disassemble a typical 12K .COM file into a 76K .ASM file containing 7500 lines of source code and a 33K cross reference file in under two minutes with 8" SD floppies. (About five times faster than others).

Two user created auxiliary files can specify labels for 8 and 16 bit values and the location of storage areas, tables and ASCII strings. The disassembled code can be sent to the console, the disk, the printer, or any combination at once.

Each package includes a 30 page manual, sample program files and variations of V-COM compatible with the TDL, MAC and ZILOG assemblers. Feature for no other disassembler at any price even comes close. ..... $80
Manual only ..... $12

## V-BUG - A Z80 Debugger

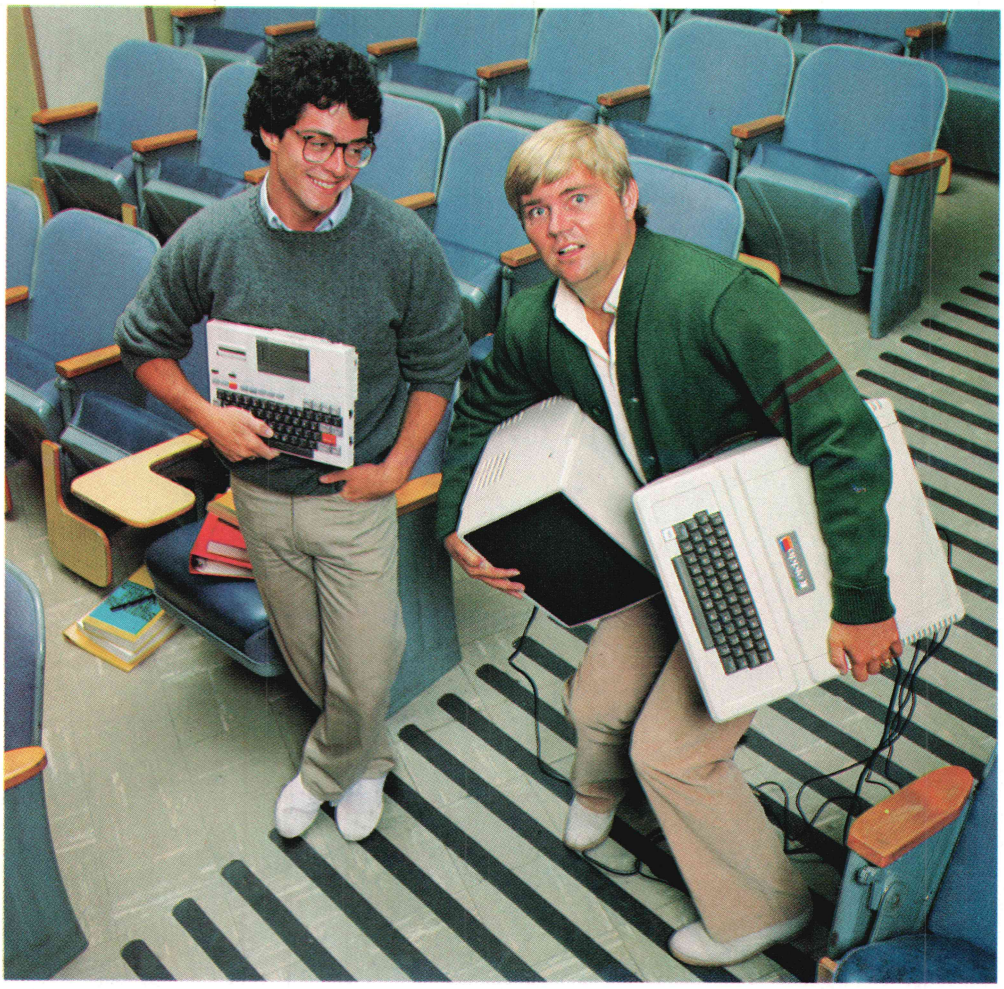
V-BUG is a combination ROM resident monitor, I/O handler and program debugger. Includes diagnostic and simple communication capability, flexible I/O assignments, CP/M compatability, complete program debugging with a disassembler and EPROM burner. Commands are specified in full words or abbreviations, allowing unlimited expandability. Intended for installation into 5K of EPROM. Complete source code. ..... $75

CompuView
PRODUCTS, INC.

Circle **no. 84** on reader service card.

1955 Pauline Blvd., Suite 200, Ann Arbor, Michigan 48103, (313) 996-1299

# ADVERTISERS INDEX



# Which do you think is the more sophisticated computer?

# Epson.

The big differences between the Epson HX-20 Notebook Computer (on the left) and the Apple Computer (on the right) are: 1) the HX-20 doesn't need a power cord, 2) the HX-20 weighs only about four pounds, and 3) the HX-20 costs a lot less money.

The Epson HX-20 Notebook Computer has a full-size keyboard, a built-in LCD screen, a built-in printer, 48K of combined RAM and ROM memory, and an internal power supply that will keep it running for over 50 hours. So you can do computing and word processing virtually anyplace you happen to be. Whereas, with the Apple Computer, you can only go as far as an extension cord will take you.

And on the HX-20, you get communications interfaces, upper and lower case letters, five program areas, a full 68 keys including an integrated numeric key pad, an internal clock/calendar, and the screen and printer. Standard. On the Apple, you pay something extra for each feature — if you can get them at all.

All of which makes the take-it-anywhere HX-20 perfect for business executives, salespeople, students, kids — anyone who's looking for an affordable, practical way into computing.

Portable. Powerful. Affordable. Sophisticated. The extraordinary HX-20 Notebook Computer. Find out just how extraordinary. Call (800) 421-5426, in California (213) 539-9140 for your nearest Epson computer dealer.

EPSON
EPSON AMERICA, INC.

Circle **no. 86** on reader service card.

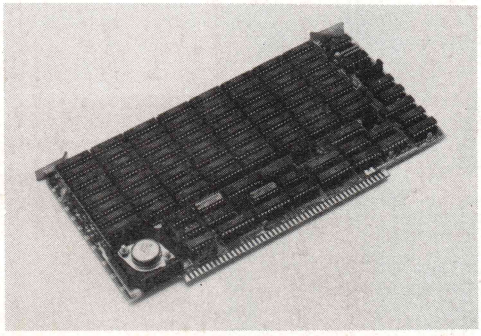
EXTRA

EXTRA

# S-100 World News

MACROTECH International Corporation 22133 Cohasset Street, Canoga Park, California 91303 • 213-887-5737

# Megabyte S-100 Memory Here Now



## Major breakthrough made by Macrotech International Corporation

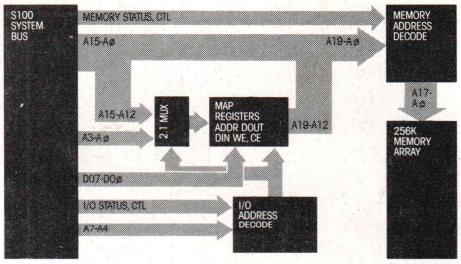
CANOGA PARK (MI)-January 20, 1983-Mike Pelkey, president of Macrotech International Corporation, today announced a major technological breakthrough in S-100 dynamic memory board density. A full megabyte of high speed dynamic ram is contained on a single standard size S-100 multilayer P.C. board. The product, dubbed 'Max' meets all IEEE/696 mechanical and electrical specifications and byte parity generation/checking is included as a standard feature. Max supports IEEE/696 24-bit addressing (selectable at any 128K boundary), 8/16 data transfer protocol, phantom line operation, and the same ultra low noise bus signal filtering provided on Macrotech's popular high performance 256K dynamic memory board.

Max is in production now and shipping at the all-time low cost per bit list price of $1,983 in unit quantity.

Bruce Kimmel, Macrotech's sales manager reports that customers are being served on a "first-in, first-out" basis and warns that due to a high incidence of graphics and similar memory-intensive applications, along with an unwillingness in the trade to pay exorbitant prices for memory, backlogs may occur for Max which could delay shipments against some late orders. With the improbability of second sourcing for some time, interested parties are urged to get orders in as soon as possible. Bruce can be contacted at 22133 Cohasset Street, Canoga Park, California 91303, or reached by telephone at (213) 887-5737.

## M³ Family Growing

Another product recently introduced by Macrotech is soaring to the top of the best-seller list. The Multiuser II is a 128 kbyte 70ns CMOS static ram memory board that is unquestionably without peer in the S-100 marketplace. It's a 6-layer board with blazing speed, 8/16 data transfer protocol, and ultra-low power external battery support. The same M³ memory mapped addressing architecture so in demand with system software professionals is now standard in the new Multiuser II. M³ was first developed by Macrotech for the popular Multiuser I 256K dynamic ram board to meet the demanding requirements of today's sophisticated systems.

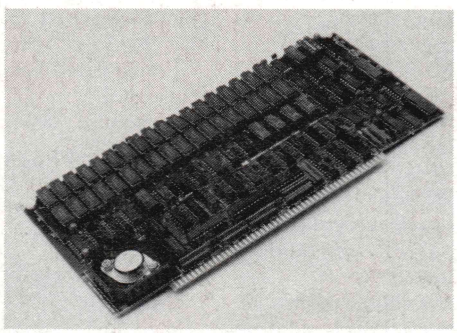



Macrotech's advanced memory mapping scheme allows each 4K block of the 16 bit (64K) logical addresses to be dynamically translated to any 4K block of the physical memory. Global memory can be configured to any size and located anywhere in the logical address space. All remaining memory can be addressed through the remaining logical address space by simply reloading the mapping registers to address the desired physical memory blocks. This scheme permits unlimited use of all on-board physical memory.

## Virtual Disk Flexibility Cited

CANOGA PARK-January 20, 1983-Macrotech reports their Multiuser I and Multiuser II S-100 ram memory boards can be used as both system memory and "virtual disk" storage in eight or sixteen-bit applications. Addressing flexibility is the key. The Multiuser M³ memory mapped addressing is guaranteed to allow memory partitioning to fit the exact requirements of your system without ever wasting a single byte.

Today's trend in operating systems appears to include extended memory capabilities to allow for the recent technological advances in semiconductor memory. A close look at Digital Research's new CP/M 3™ for example, would lead you to believe that it was especially created to fit Macrotech's family of Multiuser memory boards. (It wasn't, but try to find one that fits better.)



***Where it all started: pictured is the popular Multiuser I, Macrotech's first product. This widely used board provides 256 Kbytes of dynamic ram with 4K page memory mapping (called M³), 8/16 bit operation, 24 bit addressing and byte parity checking.***

## *MACROTECH Announces Distribution Expansion*

CANOGA PARK-January 20, 1983-Macrotech is now establishing domestic and international dealer/representative networks. The California based firm is expanding it's customer support through these channels and invites inquiries. Volume users and retailers should contact the company for details.

Macrotech's marketing director Bob Ryle states, "IEEE/696 has made S-100 legitimate. It is rapidly gaining acceptance due to its inherently superior speed characteristics." Ryle attributes the growing demand for Macrotech memories to Macrotech's strict adherence to the IEEE standard.